Southern Methodist University

SMU Scholar

Computer Science and Engineering Theses and Dissertations

Computer Science and Engineering

2020

Automating Cyber Analytics

Matthew Zaber
mzaber@smu.edu

Follow this and additional works at: https://scholar.smu.edu/engineering_compsci_etds

 Part of the Information Security Commons

Recommended Citation

Zaber, Matthew, "Automating Cyber Analytics" (2020). *Computer Science and Engineering Theses and Dissertations*. 14.
https://scholar.smu.edu/engineering_compsci_etds/14

This Dissertation is brought to you for free and open access by the Computer Science and Engineering at SMU Scholar. It has been accepted for inclusion in Computer Science and Engineering Theses and Dissertations by an authorized administrator of SMU Scholar. For more information, please visit http://digitalrepository.smu.edu.

AUTOMATING CYBER ANALYTICS:

A FRAMEWORK FOR SECURITY METRIC BENCHMARKING

Approved by:

Dr. Suku Nair

Dr. Frank Coyle

Dr. Jennifer Dworak

Dr. Liguo Huang

Dr. Jeff Tian

AUTOMATING CYBER ANALYTICS:

A FRAMEWORK FOR SECURITY METRIC BENCHMARKING

A Dissertation Presented to the Graduate Faculty of the

Lyle School of Engineering

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Science

by

Matthew Zaber

M.S, Security Engineering, Southern Methodist University
B.S., Math, College of Charleston

May 16, 2020

Copyright (2020)

Matthew Zaber

All Rights Reserved

Zaber, Matthew M.S, Security Engineering, Southern Methodist University
B.S., Math, College of Charleston

Automating Cyber Analytics:
A Framework for Security Metric Benchmarking

Advisor: Dr. Suku Nair
Doctor of Philosophy degree conferred May 16, 2020
Dissertation completed April 17, 2020

Model based security metrics are a growing area of cyber security research concerned with measuring the risk exposure of an information system. These metrics are typically studied in isolation, with the formulation of the test itself being the primary finding in publications. As a result, there is a flood of metric specifications available in the literature but a corresponding dearth of analyses verifying results for a given metric calculation under different conditions or comparing the efficacy of one measurement technique over another. The motivation of this thesis is to create a systematic methodology for model based security metric development, analysis, integration, and validation. In doing so we hope to fill a critical gap in the way we view and improve a system's security.

In order to understand the security posture of a system before it is rolled out and as it evolves, we present in this dissertation an end to end solution for the automated measurement of security metrics needed to identify risk early and accurately. To our knowledge this is a novel capability in design time security analysis which provides the foundation for ongoing research into predictive cyber security analytics. Modern development environments contain a wealth of information in infrastructure-as-code repositories, continuous build systems, and container descriptions that could inform security models, but risk evaluation based on these sources is ad-hoc at best, and often simply left until deployment. Our goal in this work is to lay the groundwork for security measurement to be a practical part of the system design, development, and integration lifecycle.

In this thesis we provide a framework for the systematic validation of the existing security metrics body of knowledge. In doing so we endeavour not only to survey the current state of the art, but to create a common platform for future research in the area to be conducted.

We then demonstrate the utility of our framework through the evaluation of leading security metrics against a reference set of system models we have created. We investigate how to calibrate security metrics for different use cases and establish a new methodology for security metric benchmarking.

We further explore the research avenues unlocked by automation through our concept of an API driven S-MaaS (Security Metrics-as-a-Service) offering. We review our design considerations in packaging security metrics for programmatic access, and discuss how various client access-patterns are anticipated in our implementation strategy. Using existing metric processing pipelines as reference, we show how the simple, modular interfaces in S-MaaS support dynamic composition and orchestration.

Next we review aspects of our framework which can benefit from optimization and further automation through machine learning. First we create a dataset of network models labeled with the corresponding security metrics. By training classifiers to predict security values based only on network inputs, we can avoid the computationally expensive attack graph generation steps. We use our findings from this simple experiment to motivate our current lines of research into supervised and unsupervised techniques such as network embeddings, interaction rule synthesis, and reinforcement learning environments.

Finally, we examine the results of our case studies. We summarize our security analysis of a large scale network migration, and list the friction points along the way which are remediated by this work. We relate how our research for a large-scale performance benchmarking project has influenced our vision for the future of security metrics collection and analysis through dev-ops automation. We then describe how we applied our framework to measure the incremental security impact of running a distributed stream processing system inside a hardware trusted execution environment.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

Figure Page

LIST OF TABLES

Chapter 1

Introduction

1.1. Motivation

Security is a cross cutting concern now more than ever. Globally connected information systems from critical infrastructure to social networks provide unprecedented access to people, things, and ideas. A key driver of this growth is the commodification of virtualization, allowing systems to scale across the world almost instantaneously. System administrators can manage the deployment and provisioning of many thousands of heterogeneous nodes through a single code base. Network engineers can verify topology changes and develop new communication protocols without interfering with production systems. Scientists can see results from large scale experiments faster and without the procurement and upkeep overhead of maintaining an in house compute cluster. The availability of near limitless global resources has had an impact on all aspects of computer science, network management, and information technology, with security being no exception.

Whether we are designing a new system from the ground up, re-architecting a legacy system for migration to the cloud, bringing a system up to regulatory compliance, or simply modernizing fleet equipment, it is necessary to define the criteria with which to measure the efficacy of the resulting solution. Often the metrics used in these decisions are based on performance or cost, with security considerations assessed during a separate compliance evaluation. In this work we consider security metrics as analogues of other system performance characteristics like network latency or CPU clock speed, with similar expectations to establish security benchmarks, sample security measurements over time, evaluate trade offs between metrics, and verify minimum security levels for a system under our control.

1.2. Modeling Security

A model is a simplified representation of some entity. As with security metrics in Section 1.3, security models can be defined and applied in different ways across the various areas of cyber security. Formal methods are a widely used technique in computer science to specify a hardware or software system as a mathematical model and verify its behaviour[47]. Attack and defense modeling go hand in hand in many[128, 132, 179, 275, 357, 358, 371, 435] threat modeling frameworks. Cyber ranges[101] are partially or fully functioning replicas of existing cyber systems built out to test attack or defense capabilities. Of particular interest in this work is Mitre's various[100] data models and enumerations of Common Weaknesses (CWE), Attack Patterns (CAPEC), Malware Attributes (MAEC), and APT group characterizations (ATT&CK) which we describe in detail later in this work.

Modeling a system's vulnerabilities and the reachability between those vulnerabilities can be found in the literature as far back as 1994 with Dacier[108] formalising the concept of privilege graphs and representing the translated graph as a Markov Model. Phillips and Swiler[315] present a separate attack graph generation method that can account for multi-stage attacks and attacker capabilities in 1998. In 1999 Ortalo[297] provides experimental results and some fundamental metrics using Markov analysis with Dacier's privilege graphs, and in 2002 Sheyner[369] describes how attack graph construction and analysis can be automated. In 2006 Ou[300] provides an analysis of scalability extensions to the MulVal[301] attack graph engine presented the previous year, and in 2013 Hong[175] presents further scalability improvements to MulVal using logic reduction techniques.

Attack graphs show the relationships among vulnerabilities within a system and provide context to security scans already conducted by many organisations. An attack graph is a directed graph that captures all possible paths an attacker can traverse within a system to reach a desired target state. The first node in the graph represents the origin of the attack and the final node denotes the target. The origin contains only outbound edges and the target contains only inbound edges. Nodes in the graph between the origin and target represent discrete states in states network. Each edge in the graph identifies a possible pivot from one

state to another through either unaltered access mechanisms or successful exploitation of a vulnerability. The conditions necessary for successful compromise of the vulnerability are encapsulated in the attack graph vertices, and include information such as network, port, protocol, and access privilege level restrictions, as well as the effect of a successful exploit on the system such as privilege escalation or remote code execution. These conditions can be populated from the output of IA and network management systems, or in hypothetical cases, can be defined manually.

1.3. Measuring Security

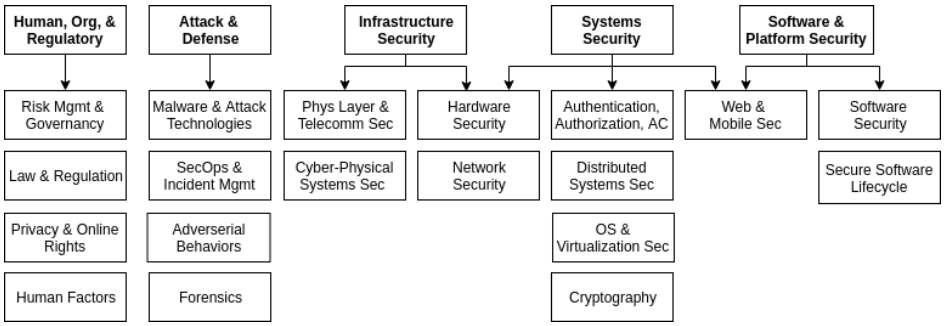


Figure 1.1: Cyber Security Body of Knowledge - Key Areas [329]

Cyber security as a subject is at once broad and deep, covering a wide range of functional topics and skill sets. The lens through which a database administrator views security differs from that of a network engineer or an application developer within a particular business, and these perspectives may diverge from corresponding roles in other industry sectors. Thus, to effectively communicate any security objectives, it is necessary to understand the frame of reference from which they arise. In this work we refer to the Cyber Security Body of Knowledge[329] (CyBoK) Key Areas(KAs) as waypoints while navigating various aspects of security. The CyBoK project, sponsored by the UK's National Cyber Security Centre, provides both a taxonomy of cyber security topics, and a mapping to canonical references for these topics. Figure 1.1 summarizes the 19 top level KAs and 5 broad categorical groupings. Rather than delve into all the different CyBoK headings here in isolation, we elect instead to refer to the relevant classes as we encounter them in developing the thesis.

Cyber security can be considered in terms of adversaries, along with their goals, and defenders, whose implicit goal is to prevent an adversary from achieving theirs. The purpose of a security metric is to quantify a specific aspect of cyber security, such as attacker capabilities or defensive controls. We now frame the problem this work addresses by summarizing the security metrics and measurement techniques currently available, armed with the vocabulary and taxonomy needed to analyze them in context.

Regulatory and Compliance Metrics: The left containing bracket in Figure 1.1 depicts Human, Organizational, and Regulatory Aspects, and within the Risk Management KA[69] we find the topic of security metrics. Indeed, security metrics are often defined within the scope of regulatory compliance. NIST's *Security Metrics Guide for Information Technology Systems* [396, 95], ISO 27004 *Monitoring, measurement, analysis and evaluation*[7] and the Center for Internet Security's *CIS Controls Measures & Metrics*[99] define the requirements for security metrics programs within various industry and government organisations. These security metrics are generally compliance focused, capturing statistics or percentages like employee training completion rate, number of hosts scanned and vulnerabilities found, or system performance complaints since the last patch roll out.

Operational Metrics The right containing bracket in Figure 1.1 labelled Attacks & Defences includes security metrics listed in the *Security Operations and Incident Response*[119] KA. The measurement processes in the operational area are distinguished primarily through automation support. Security Information and Event Management (SIEM) systems play a large role in collecting and aligning system telemetry for incident monitoring and response. SIEMs provide correlation of host/network event logs, IDS/IPS alerts, threat/vulnerability feeds, etc, and present a unified view of the system's security posture automatically to the SOC. Before the advent of managed SIEMs, system administrators typically filled the role of security engineers, and relied on hand rolled collections of shell/perl scripts to manage systems, parse logs, collect or push events, format reports, and issue alarms. To be effective required tribal knowledge along with proficiency in programming, network plumbing, and systems management, so changes to the environment or workforce made it extremely

difficult(expensive) to deliver continuous monitoring capabilities to operators at any scale.

Vulnerability Metrics Common to the 3 interior categories of Infrastructure, Systems, and Software & Platform Security in Figure 1.1 are vulnerability metrics. The Common Vulnerability Scoring System[262] (CVSS) is an open framework used throughout government and industry to report the severity of specific security vulnerabilities in commonly used software products. CVSS scores range from 0 to 10 based on the vulnerability's exploitability and impact, with a score of 10 signifying the highest severity. Exploitability is calculated by determining the access vector, access complexity, and number of authentication attempts required to exploit the vulnerability, with higher exploitability values equating to an easier compromise. Impact scores are determined by identifying the scope of a successful exploit on the vulnerable system's confidentiality, integrity, and availability.

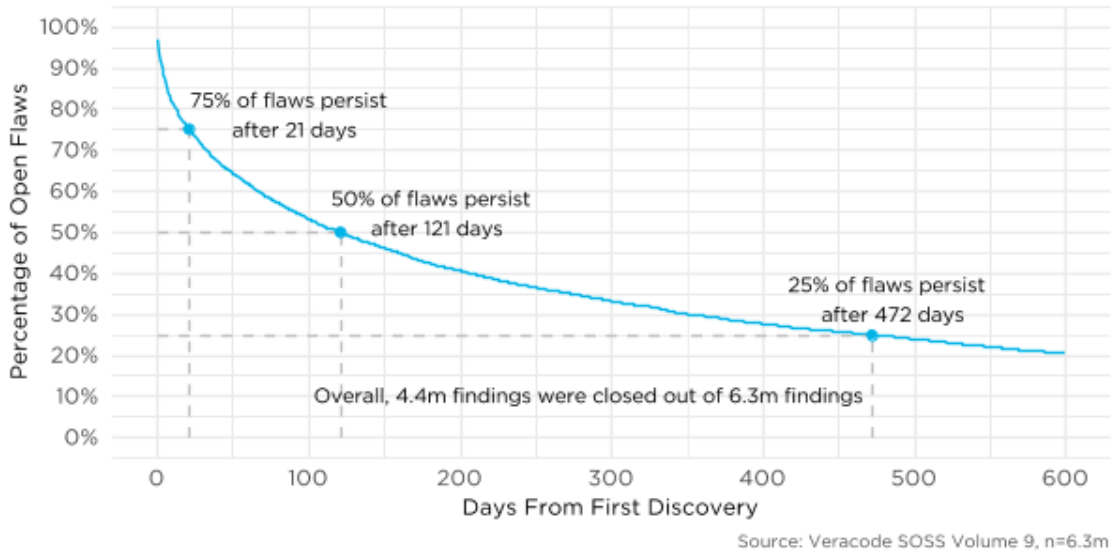


Figure 1.2: Veracode Static Analysis Remediation Times SOSS [385]

Static analysis[43] is a common method to identify possible vulnerabilities in software not included in the CVSS set. Developers can automatically scan custom source code and 3rd party libraries from within their IDE, or include hooks in their repository/CI to check incoming commits before merging. The metrics produced from static analysis can alert developers[194] to common vulnerabilities in the code they write. Figure 1.2 shows the survival rate for static scan findings in 2018.

Economic Metrics Measuring the trade-offs between competing interests and the incentives influencing actors is pervasive throughout cyber security. Security economics[31] applies prevailing microeconomics models to describe limiting factors for attack and defense agents. The Gordon-Loeb[152] model details an optimal security investment under the assumption of diminishing marginal returns. In *Security Metrics and Security Investment*[59] the authors derive metrics including the return on security investment (ROSI) and net present value (NPV) over future periods while accounting for different risk management temperaments. A survey of the economics of information security[33] literature summarizes potential applications of these economic metrics in many of the key cyber security areas.

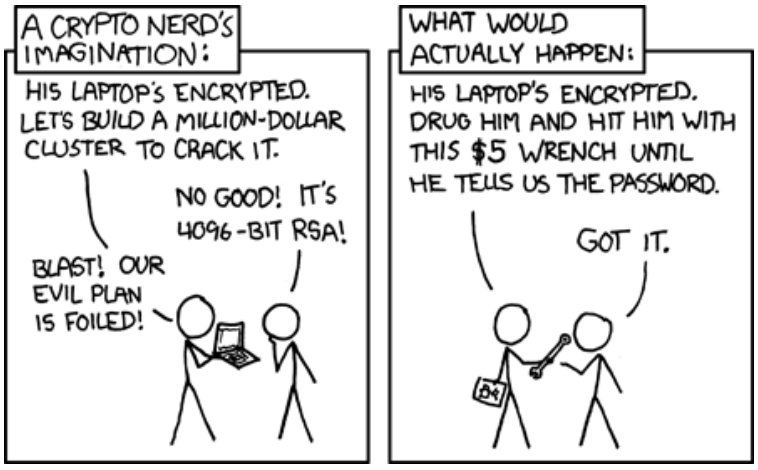


Figure 1.3: source: https://www.xkcd.com/538/

Figure 1.3 shows anecdotally one type of situation that economic metrics help us to avoid. Given an existing defensive capability that already deters attacks (data at rest cryptography in this case), further investment in that defense will give limited additional value. It also demonstrates the dangers of evaluating risk based on too narrow a security domain. Defining an attacker's capabilities, or threat modeling, is described in Section 1.2.

The evolution of information systems is trending away from a single administrative domain with a clear security boundary and moving towards a mix of self-hosted, provider managed, and remote 3rd party services. This creates unique challenges from an information security perspective, not least of which being how to define the new perimeter.

1.4. Automating Security Measurement

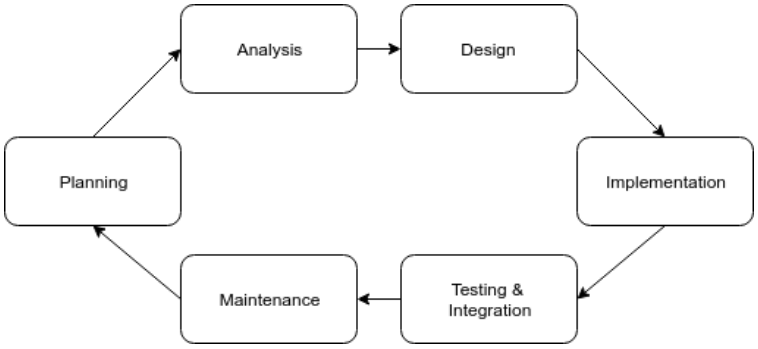


Figure 1.4: Software Development Life Cycle

The software development life cycle shown in Figure 1.4 is an iterative process that, until recently, was almost exclusively used within the Software & Platform KA. Before the Agile[45] method was adopted, a single iteration of the SDLC could span several years, with planning, design, and analysis consuming a significant portion of the up front time and cost for the project. When a software project failed under this model, it happened because available funding was burned and often resulted in little or no working code. The Agile approach prefers faster iterations (often counted in weeks) with less emphasis on up front specifications. The result is a 'fail-fast' model that surfaces fundamental problems early and produces prototype software that may be salvaged and reused in other work if the project is abandoned.

An ecosystem of tools exists around the SDLC that are mature, actively maintained, and most importantly (for us at least) almost entirely automated. Design requirements are captured in trackers that contain customizable metadata like priority, status, and dependencies. These requirements can link to code commits or PRs that satisfy the requirement, or customer created issues or development blockers that prevent the requirement from being closed. Unit, Integration, Regression, and Acceptance tests are run by continuous integration and delivery systems which can prevent breaking changes from entering the code base, roll back bad commits that do get merged, and deploy updates to production servers as dictated by the pipeline configuration.

As an extension (and tacit endorsement) of the automated software CI/CD pipeline described above, development operations(DevOps) has recently emerged as the corollary in managing the life cycles components in the System and Infrastructure CyBOK categories. DevOps allows systems and network engineers to develop, test, deploy, and provision the supporting infrastructure required by an application in the same workflow, even from the same code base as the desired application. Maintaining the desired state of the underlying system in configuration files and provisioning scripts through a version control system is often called *Infrastructure-as-Code*, While this is sometimes used to set up bare metal resources like servers and network appliances, the more common case currently is to provision infrastructure onto a self managed or commercial hypervisor using virtual machines or containerized micro services.

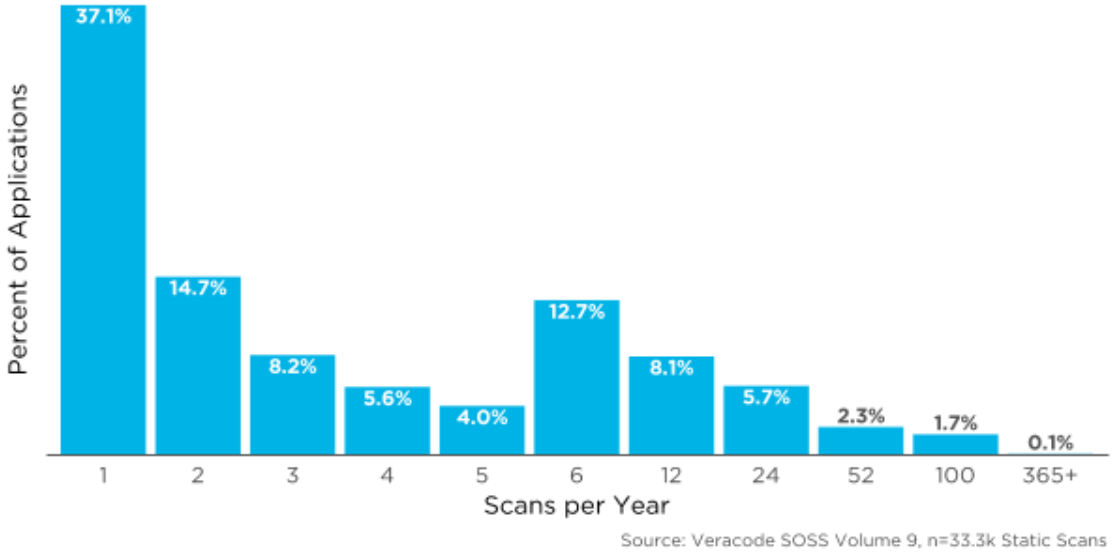


Figure 1.5: Veracode Freq Percentages SOSS [385]

A functioning DevOps workflow integrates the application development and operations teams. The next logical step in this evolution is to disentangle the stovepiped security evaluation mechanisms built around each of these formerly independent areas. The resulting field of SecDevOps (or DevSecOps[408]) is *not* the sum of the security of each part (although it will likely be treated that way at least until the community converges on which term to reference it by). Rather, it should trigger a re-assessment of previous security controls and

best practices prescribed through existing regulatory and compliance frameworks.

For example, Figure 1.5 shows the yearly count from over 33 thousand source code scans in 2018 as reported by Veracode. While this data is from a single vendor, the results are still concerning. The median number of security scans per year was just 2. We presume there is regulatory or compliance pressure to scan at least once per year. Given the percentage of applications that scan once a week or more is just over 4%, we can also infer that most customers either don't have an automated development workflow in house, or if they do have chosen not to integrate this capability.

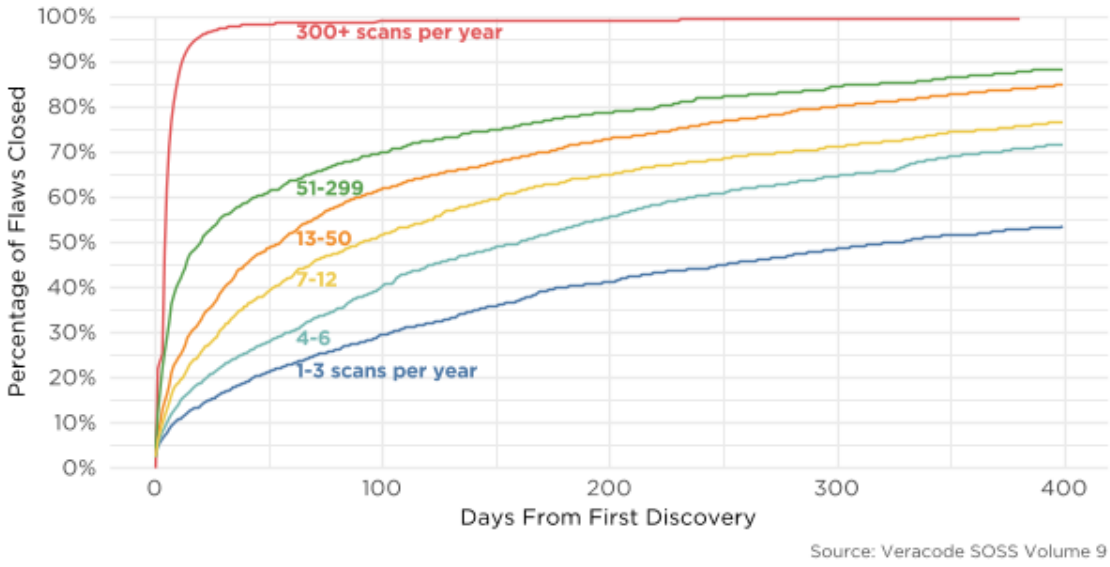


Figure 1.6: Veracode Scan Frequencies vs Closures SOSS [385]

While we can't extrapolate with confidence the closure rates in Figure 1.2 to CVSS findings, or the scan rates in Figure 1.5 to users of alternative static analysis products, we can draw a very clear line between the teams that scan their code as part of an automated development workflow and their ability to address security findings. Figure 1.6 displays a distinct stratification of response times based on scan frequency. Indeed, the teams that incorporated a simple static analysis check into their DevOps pipeline were able to remediate nearly all found flaws before the others could fix half. What we observed from Figure 1.2, that around half the observed samples only conducted one or two scans per year suggests that, in this context, they were only able to fix about 50% of the identified flaws after more

than 400 days. Referring back to the Agile development principals that DevOps workflows are based upon, we can surmise the relative speed in addressing flaws is attributable to small incremental changes with actionable feedback after each change.

1.5. Contributions

In this thesis we provide a framework for the systematic validation of the existing security metrics body of knowledge. In doing so we endeavour not only to survey the current state of the art, but also to create a common platform for future research in the area to be conducted.

We then demonstrate the utility of our framework through the evaluation of leading security metrics against a reference set of system models we have created. We investigate how to calibrate security metrics for different use cases and establish a new methodology for security metric benchmarking. To support automated and repeatable testing, each metric is implemented as an independent module containing the computation logic. These metric implementations can then be called directly from an embedded library, accessed through a benchmark test with standard interfaces and return types, or managed with a container orchestration engine. Each test has an associated specification which is used to configure dependencies and tuning parameters for execution. Specifically, we:

- Provide a methodology for calibrating security metrics against a fixed baseline and demonstrate isolation criteria for benchmark regimens
- Implement several distinct execution paradigms for different use cases
- Describe validation mechanisms available through increasingly sophisticated modeling and simulation extensions

We further explore the research avenues unlocked by automation through our concept of an API driven S-MaaS (Security Metrics-as-a-Service) offering. We review our design considerations in packaging security metrics for programmatic access, and discuss how various client access-patterns are anticipated in our implementation strategy. Using existing metric processing pipelines as reference, we show how the simple, modular interfaces in S-MaaS

support dynamic composition and orchestration. We anticipate consumers will fall into one or more of the following access patterns:

- **R&D**: A clean environment for developing and testing metrics.
- **Reporting**: Measurements are pushed to persistent storage or pulled from a client into a structured format for charting and analysis.
- **Batch**: Client asynchronously submits a collection of inputs to calculate metrics for, with the target use-case being dataset preparation for supervised learning.
- **Stream**: Client defines dependency graphs amongst metrics and connects these topologies to data sources for continuous stream processing.

Next we review aspects of our framework which can benefit from optimization and further automation through machine learning. First we create a dataset of network models labeled with the corresponding security metrics. By training classifiers to predict security values based only on network inputs, we can avoid the computationally expensive attack graph generation steps. We use our findings from this simple experiment to motivate our current lines of research into supervised and unsupervised techniques such as network embeddings, interaction rule synthesis, and reinforcement learning environments.

Finally, we examine the results of our case studies. We summarize our security analysis of a large scale network migration, and list the friction points along the way which are remediated by this work. We relate how our research for a large-scale performance benchmarking project has influenced our vision for the future of security metrics collection and analysis through DevOps automation. We then describe how we applied our framework to measure the incremental security impact of running a distributed stream processing system inside a hardware trusted execution environment.

This work begins to develop the foundation for a repeatable, extensible security analytics benchmarking framework, allowing existing and proposed metrics to be implemented in uniformly accessible ways and tested against standard reference models. Through au-

tomation we lower the barrier of entry for future research and expand accessibility to client applications not currently integrating security metrics in practice.

Chapter 2

Background

Aristotle is often credited with founding the fields of natural science in the West; with the story going that he disagreed with his teacher Plato on the matter of philosophical thought, arguing that it should be based on observations from the natural world. We consider observations to be *measurements* when they are quantified with respect to an agreed upon scale, or measurement unit. The first measurement units were used in bartering to standardize the exchange values of different commodities[276]. To measure length, for example, the torso was consistent enough across people to be considered a measurement unit, from which other units like the hand, the foot, and the cubit were derived. A *quantity* refers specifically to a property that can be measured, in this case length. Although the term is often[118] overloaded to mean the amount of something as well, we attempt to be metrologically consistent in this work by referring to the property to be measured as the quantity or *measurand*, and the *amount* of some thing as the observed value in terms of magnitude and measurement unit.

In recent years effort has been spent investigating the science of cyber security[212, 355, 361, 381]. In the *NSF/IARPA/NSA Workshop on the Science of Security*[134] three directions of research were identified to improve the scientific foundations of security, *metrics*, *formal methods*, and *experimentation*. In this thesis we focus particularly on the role of metrics in security research, although well-defined system models and experimentation environments both play key roles in the analysis. In this chapter we review the relevant concepts in security metrics needed to build our framework. Section 2.1 describes methods for representing different components influencing security. These models are the inputs to the security property being measured, and in Section 2.2 we describe the types of metrics available and how they relate.

2.1. Security Modeling

An argument can be made that the accuracy of any measurement taken from a model is limited, and that the only true test of security is in measuring the production system. While it is possible to test systems in production, these tests can be disruptive to operations, for example when testing fire alarms in a building or conducting social engineering on customers or staff. We can recreate the production system on a cyber range[449] or similar dedicated environment for security testing, without the advantages of or impact on live traffic. In many cases we can interrogate the production system for values from which to build a model, but only if the system is in production. Design time tools are abundant throughout the CyBOK KAs, but the backend models used by tools differs in schema and accessibility, making integration with security metrics difficult. The role models play varies widely depending on the area of cyber security, with intrusion detection and antivirus systems on one end of the spectrum requiring production traffic for testing and evaluation, and cryptographic protocols on the other end having model based proofs of their security properties (although implementation is a different matter). The regulatory and compliance frameworks listed in Section 1.2 are made up of individual security controls, each of which can fall somewhere on this empirical spectrum. The use of models may be overly naive for some cases, but where applicable, models enable programmatic access and rapid prototyping, and allow us to isolate specific security properties for measurement through simulation and analysis at a speed not possible with live systems. In the rest of this section we provide some useful models that we make use of in later chapters.

2.1.1. Infrastructure & System Modeling

The OSI 7-layer model[465] describes network communication as a stack of protocols built on top of a *Physical* transport medium such as copper, optical fiber, or radio. On top of the physical layer are the Layer 2 *Data Link* protocols that facilitate reading and writing frames to and from the given physical circuit including source and destination addressing (*Medium Access Control*), and the protocols that encapsulate/de-encapsulate upper layer packets into

the frame format suitable for transmission across the medium (*Logical Link Control*). When a datagram from the upper layers is transmitted to a remote system, it is encapsulated into one or more packets at the *Network* layer, the packets are then encapsulated into frames at the Data Link Layer, and the frames are placed onto the physical medium. When a frame is received at the destination, it is de-encapsulated and the payload is handed up to the Layer 3 *Network* protocol capable of processing the packet. Encapsulation or de-encapsulation occurs between each layer in the OSI model and adds processing overhead and increased latency to traffic. Layer 2 traffic is not routable, since routing information like IP address is stored in the Layer 3 header.

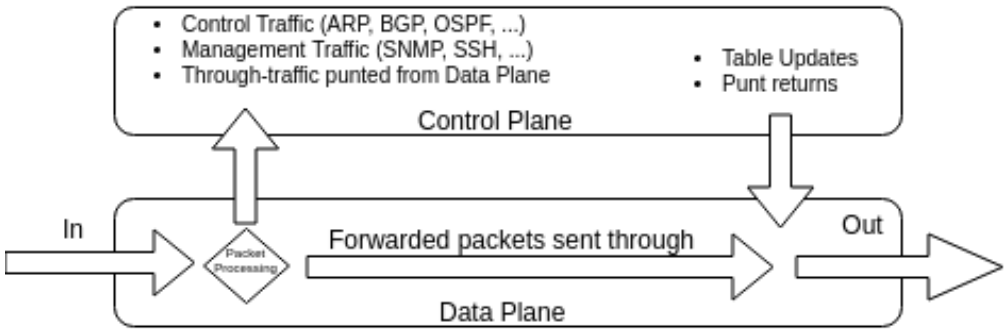


Figure 2.1: Traffic flow between planes

Network device traffic can be partitioned into two planes[202]: the *Forwarding (or Data) Plane* consists of traffic passing through the device while *Control Plane* traffic has the network element as its destination. We consider the *Management Plane* a subset of the *Control Plane*. Forwarding plane components do the heavy lifting in a network device by connecting incoming traffic to the appropriate outgoing port. When the forwarding element can't identify the correct egress port the packet is sent up to the control plane for a routing solution as shown in Figure 2.1. The control plane can request and process information about link status and network topology from peers and update forwarding tables in the data plane in response.

In conventional network equipment the control and data planes are tightly coupled. That is, the components responsible for forwarding packets are physically located along side the

components used to make routing and switching decisions about that traffic. How these systems are laid out in practice varies widely by function and vendor. Figure 2.2 depicts a notional router. Line cards house the forwarding engine in Application Specific Integrated Circuits (ASICs), Network Processing Units (NPUs), or even software deployed on virtual machines. Incoming traffic is read off the port into a buffer, the packet header is examined to query the Forwarding Information Base for the destination, the packet processor alters the header as needed, and the packet is forwarded out the appropriate egress port. If the traffic is destined for the device (that is, control or management plane) or if the FIB lookup failed, the packets are queued in the receive path buffer for transfer to the routing engine. The routing engine is typically housed on another line card and connected to the forwarding modules through the backplane or switch fabric. Control plane services are hosted by the network OS running on commodity CPUs. The primary purpose of the control plane service suite is to manipulate the forwarding plane lookup tables based on routing and signalling information it receives. It maintains the Routing Information Base (RIB) used to optimize local FIBs and provides the interface for router management and configuration.

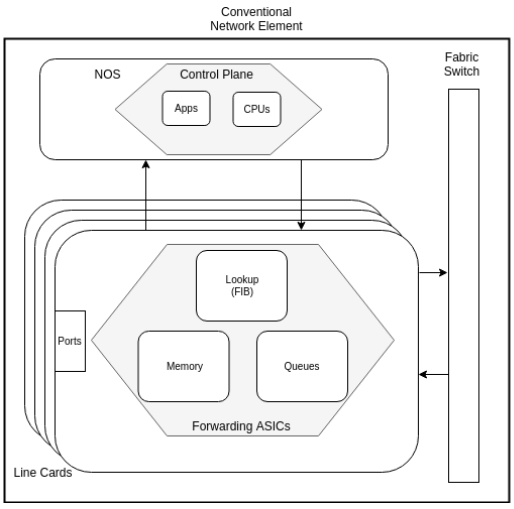


Figure 2.2: Conventional Element

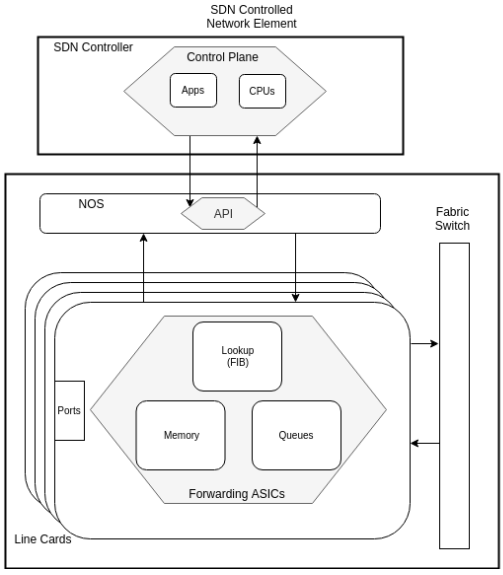


Figure 2.3: SDN Controlled Element

SDN systems decouple the forwarding and control plane elements further, by centralizing controller functionality outside the data plane's enclosure and introducing a southbound

interface with which to alter forwarding tables via API. This allows SDN controllers to scale independently of the forwarding elements and receive state information for all network devices it oversees. Since the complex decision logic has been extracted from the forwarding units, the core elements in SDN are simple switches. The result is a control plane that can optimize the forwarding rules it pushes based on global requirements instead of the limited view available to conventional devices. This also results in a potentially larger attack surface and single point of failure.

Carrier network architectures can be decomposed into tiers based on functionality and proximity to the end users. The *Access* tier is the forward facing attachment point for a user such as the radio tower a mobile device connects with or the set top box that joins a home network to the ISP. The *Provider Edge* provides the services needed to manage traffic between the various access platforms and the provider core. One or more *Aggregation* layers can be added to consolidate provider edge points based on access density requirements. Finally, the *Provider Core* provides hi speed transit between edge nodes.

2.1.2. Threat Modeling

The intrusion kill chain introduced by Lockheed-Martin[179] describes the steps an actor takes in attacking a target system. The specifics of each step vary by situation and objective, but the process provides a foundation on which to build and compare models.

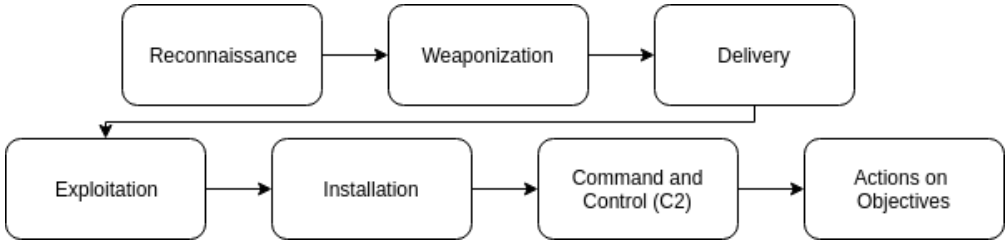


Figure 2.4: LM Intrusion Kill Chain[179]

Attacks against networks can be motivated by many factors, but the effects generally fall into one of a handful of categories[84]. Our threat model assumes an attacker originating outside the core network boundary intends to target a device within the core network,

permitting the attacker to view or alter a victim's traffic. In other words, identifying the potential for *eavesdropping* and *tampering* is the focus of this research, and we address the limitations of modeling network *disruption* later in this section. Table 2.1 lists some common attack patterns against Layer 2 and 3 networks. In general these attacks are used to redirect a target's traffic through an asset controlled by the attacker, or to escalate the attacker's privileges on systems that interact with the target. We briefly review these attack classes here in order to present our Datalog models in the following sections.

Table 2.1: Potential Threats

VLAN Hopping	ACL Bypass
STP Injection	BGP Hijacking
ARP Cache Poisoning	Route Table Poisoning
MAC Flooding	SYN Flooding
CAM Overflow	Packet Crafting
MAC/DHCP Spoofing	IP Spoofing

We frame our threat discussion within Layers 2 and 3 of the OSI 7-layer model here for clarity, but don't limit our analysis to only these vectors. These attacks often aren't the goal of an attacker, but rather enable the attacker to reach their goal through the effects of the compromise. For example, traffic eavesdropping may reveal credentials for a privileged account on some other system the attacker has interest in.

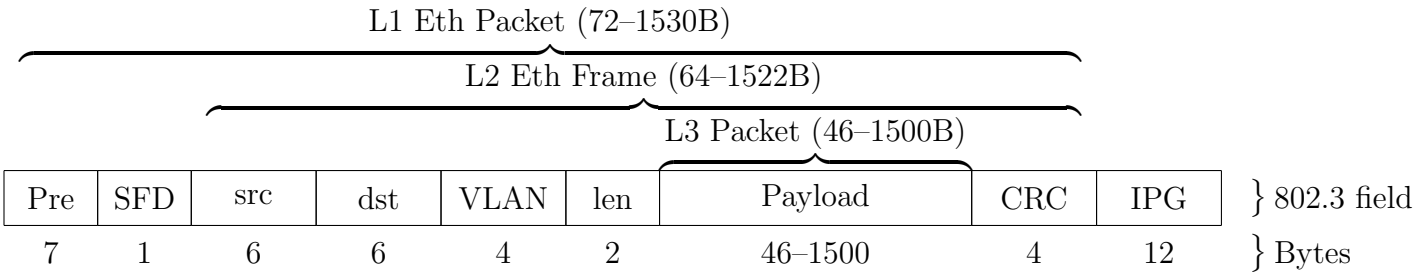


Figure 2.5: 802.3 Ethernet Packet Layout

Layer 2 Attacks:

- CAM Overflow: Hardware switches use *Content Addressable Memory* tables to map devices to the port they are connected to. When frames enter the ingress switch port,

a CAM table entry is created for the source MAC address and port if it doesn't exist. The destination MAC address is found in the CAM table and the frame is forwarded out the associated port. If no destination entry is found the frame is flooded out all ports, and the CAM table is updated with the port the destination MAC responds from. The CAM is implemented in hardware and has a fixed size buffer. If an attacker can exhaust the CAM buffer by generating enough unique source addressed frames to fill the table, the switch will flood any traffic without an entry to every switch port, allowing an attacker to eavesdrop traffic from any connected device on the native VLAN.

- ARP Spoofing: The *Address Resolution Protocol*[317] maps Layer 2 MAC addresses to Layer 3 IP addresses. To find a MAC address for a given IP target, an ARP request is broadcast to all members of the local subnet. The owner of the IP address then sends an ARP reply containing their MAC address. RFC 826 allows for unsolicited ARP replies, meaning that any system on the local subnet can announce that they own any IP or MAC address without the address first being requested by a peer. An attacker announcing ownership of an address will receive all traffic destined for that address.

- VLAN Hopping: *Virtual LANs* allow the creation of multiple logically separate broadcast networks over the same Layer 2 switch. The 802.1Q VLAN field in Figure 2.5 accepts a 12 bit VLAN tag to differentiate 4094 possible VLANs, with later extensions to the standard allowing over 16 million tags. Since VLANs are isolated broadcast domains, communication between VLAN nodes must be routed over a Layer 3 protocol. VLAN tags are written to Ethernet frames by the ingress switch either statically based on attachment port, or dynamically based on some policy like the source MAC address. The primary VLAN connection types are *Access* links which connect a host to a switch using a single tag, *Trunk* links which interconnect switches and carry tags for all VLANs, and the default *Native* VLAN which allows untagged traffic and is typically used for management. The link type is configured for each switch port man-

ually over the management interface, dynamically using a protocol like DTP (dynamic trunking protocol), or it falls back to the default which is vendor and model specific. If a switch port the attacker is connected to is not explicitly configured as an Access port, the attacker can present themselves as a peer switching device (known as *Switch Spoofing*) and craft the DTP packets to negotiate a Trunk port, resulting in access to traffic on any managed VLAN. Another VLAN bypass can be accomplished if the attacker is allowed to write arbitrary data to the VLAN tag field, which is permitted by members of the Native VLAN. In this scenario, an attacker can send traffic to a target on another VLAN by *Double Tagging* messages. The outer VLAN tag is popped by the first receiving switch and then flooded out all of its Native VLAN ports. The inner tag is now read by the receiving switch and the message is sent to the victim. This is a blinded attack since no response will be returned from the malicious traffic.

2.1.3. Graph Based Models

Vulnerability exploitation can be thought of as a set of preconditions which must be true for an attack to be successful, and a set of postconditions that become true after a vulnerability is successfully exploited. In this context, an attack graph is a tool that can efficiently evaluate the set of preconditions that exist in a system and determine if attaining a specific postcondition is possible. When an attack goal is reachable in a system, the attack graph will enumerate all sequences of preconditions necessary to assert the goal's postcondition is true. When the goal is not reachable, no attack graph is produced. To date, research in this area has focused on software vulnerabilities. A likely reason being that a large amount of data is collected and publicly disseminated in the form of the National Vulnerability Database. The NVD provides a number of metrics associated with each vulnerability along with metadata like access vector and potential effects.

Modeling a system's vulnerabilities and the reachability between those vulnerabilities can be found in the literature as far back as 1994 with Dacier[108] formalising the concept of privilege graphs and representing the translated graph as a Markov Model. Phillips and

Swiler[315] present a separate attack graph generation method that can account for multi-stage attacks and attacker capabilities in 1998. In 1999 Ortalo[297] provides experimental results and some fundamental metrics using Markov analysis with Dacier's privilege graphs, and in 2002 Sheyner[369] describes how attack graph construction and analysis can be automated. In 2006 Ou[300] provides an analysis of scalability extensions to the MulVal[301] attack graph engine presented the previous year, and in 2013 Hong[175] presents further scalability improvements to MulVal using logic reduction techniques. In 2015 Abraham[11] introduces the Cyber Security Analytics Framework(CSAF) which we adopt for this analysis. More thorough surveys of the canonical attack graph literature can be found in [209] and [237].

Attack graphs show the relationships among vulnerabilities within a system and provide context to security scans already conducted by many organisations. An attack graph is a directed graph that captures all possible paths an attacker can traverse within a system to reach a desired target state. The first node in the graph represents the origin of the attack and the final node denotes the target. The origin contains only outbound edges and the target contains only inbound edges. Nodes in the graph between the origin and target represent discrete states in states network. Each edge in the graph identifies a possible pivot from one state to another through either unaltered access mechanisms or successful exploitation of a vulnerability. The conditions necessary for successful compromise of the vulnerability are encapsulated in the attack graph vertices, and include information such as network, port, protocol, and access privilege level restrictions, as well as the effect of a successful exploit on the system such as privilege escalation or remote code execution. These conditions can be populated from the output of IA and network management systems, or in hypothetical cases, can be defined manually.

The example in Figure 2.6 assumes an attacker located on the public internet with the target being root access on the internal workstation. The network model provided to MulVal is shown in Listing 2.1. The *hacl*/4 clauses define access control rules that depict router and firewall configurations and dictate reachability between states. The host configuration

properties include what services are running, the user or privilege level that service is running as, and any vulnerabilities known to affect that service.

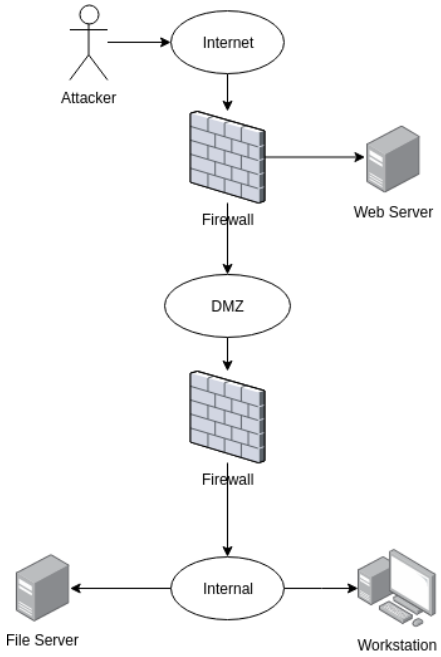


Figure 2.6: Example network

```
1  attackerLocated(internet).
2  attackGoal(execCode(workStation,_)).
3  hacl(internet, webServer, tcp, 80).
4  hacl(webServer, _, _, _).
5  hacl(fileServer, _, _, _).
6  hacl(workStation, _, _, _).
7  hacl(H,H,_,_).
8  /* configuration information of fileServer */
9  networkServiceInfo(fileServer, mountd, rpc,
       100005, root).
10 nfsExportInfo(fileServer, '/export',
       _anyAccess, workStation).
11 nfsExportInfo(fileServer, '/export',
       _anyAccess, webServer).
12 vulExists(fileServer, vulID, mountd).
13 vulProperty(vulID, remoteExploit,
       privEscalation).
14 localFileProtection(fileServer, root, _, _).
15 /* configuration information of webServer */
16 vulExists(webServer, 'CAN-2002-0392', httpd).
17 vulProperty('CAN-2002-0392', remoteExploit,
       privEscalation).
18 networkServiceInfo(webServer ,httpd, tcp ,80
       ,apache).
19 /* configuration information of workStation
       */
20 nfsMounted(workStation, '/usr/local/share',
21   fileServer, '/export', read).
```

Listing 2.1: input.P [300]

The visual representation of the resulting attack graph can be found in Figure 3.9a. A MulVal attack graph contains three types of nodes: **AND**, **OR**, and **LEAF**. *LEAF* nodes (rectangles) describe known facts like configuration information and attacker privilege that are given as inputs to the system model. Internal nodes generally represent potential privileges to be gained by an attacker. *AND* nodes (ovals) contain interaction rules that dictate which facts and conditions are necessary to derive new knowledge. *OR* nodes (diamonds) represent derived facts such as transition states possible given all incoming conditions are satisfied.

A brief reading of the attack graph generated in Table 2.2 and Figure 3.9a from the top

left:

- An attacker from the internet (node 18) can access *webServer* on port 80 (node 15)
- Vulnerability CAN-2002-0392 can be exploited (node 20) to allow remote code execution on *webServer* as user *apache*. From here the attacker can either (node 13):
  1. Access *fileServer* directly over rpc (node 10) and use this to escalate privilege to root (node 22) and write malicious files served to *workstation* (node 5).
  2. *OR* the attacker can write malicious files served to *workstation* directly using NFS shell (node 23)
- Given *workstation* can access the malicious file (node 26) and the malicious file has been created (node 5) then *workstation* can execute the malicious file with *root* privileges allowing the attacker to achieve the target.

Table 2.2: MulVal gernerated output

ID	Text	Type	Leaf
1	execCode(workStation,root)	OR	0
2	RULE 4 (Trojan horse installation)	AND	0
3	accessFile(workStation,write,'/usr/local/share')	OR	0
4	RULE 16 (NFS semantics)	AND	0
5	accessFile(fileServer,write,'/export')	OR	0
6	RULE 10 (execCode implies file access)	AND	0
7	canAccessFile(fileServer,root,write,'/export')	LEAF	1
8	execCode(fileServer,root)	OR	0
9	RULE 2 (remote exploit of a server program)	AND	0
10	netAccess(fileServer,rpc,100005)	OR	0
11	RULE 5 (multi-hop access)	AND	0
12	hacl(webServer,fileServer,rpc,100005)	LEAF	1
13	execCode(webServer,apache)	OR	0
14	RULE 2 (remote exploit of a server program)	AND	0
15	netAccess(webServer,tcp,80)	OR	0
16	RULE 6 (direct network access)	AND	0
17	hacl(internet,webServer,tcp,80)	LEAF	1
18	attackerLocated(internet)	LEAF	1
19	networkServiceInfo(webServer,httpd,tcp,80,apache)	LEAF	1
20	vulExists(webServer,'CAN-2002-0392',httpd,remoteExploit,privEscalation)	LEAF	1
21	networkServiceInfo(fileServer,mountd,rpc,100005,root)	LEAF	1
22	vulExists(fileServer,vulID,mountd,remoteExploit,privEscalation)	LEAF	1
23	RULE 17 (NFS shell)	AND	0
24	hacl(webServer,fileServer,nfsProtocol,nfsPort)	LEAF	1
25	nfsExportInfo(fileServer,'/export',write,webServer)	LEAF	1
26	nfsMounted(workStation,'/usr/local/share',fileServer,'/export',read)	LEAF	1

(a) VERTICES.csv

src vertex	dst vertex	weight
6	7	-1
11	12	-1
16	17	-1
16	18	-1
15	16	-1
14	15	-1
14	19	-1
14	20	-1
13	14	-1
11	13	-1
10	11	-1
9	10	-1
9	21	-1
9	22	-1
8	9	-1
6	8	-1
5	6	-1
23	24	-1
23	25	-1
23	13	-1
5	23	-1
4	5	-1
4	26	-1
3	4	-1
2	3	-1
1	2	-1

(b) ARCS.CSV

Along with the visualization in Figure 3.9a, MulVal also produces comma separated value

formatted output which lends itself more readily to further analysis. The lists of edges and vertices in Table 2.2 describe the attack graph generated from Listing 2.1.

2.1.4. Summary

2.2. Security Metrics

NIST 800-55[396] describes security metrics as "*tools designed to facilitate decision making and improve performance and accountability through collection, analysis, and reporting of relevant performance-related data. IT security metrics must be based on IT security performance goals and objectives.*" This section reviews some of the available security metrics from the literature loosely grouped by each Cybok heading, along with properties that have been used to characterize these metrics.

2.2.1. Existing Security Metrics Taxonomies

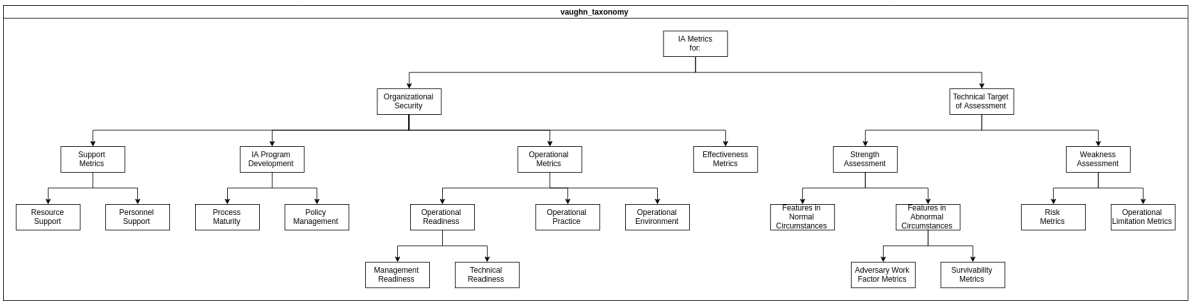


Figure 2.7: Vaughn's Security Metric Taxonomy[411]

Vaughn's taxonomy[411] from 2003 is heavily influenced by federal, and in particular defense department, perspectives on information assurance metrics. The classification tree is heavy on the side of personnel and regulatory metrics compared to other surveys, and the categories draw from military concepts of operational readiness, threat identification, and target acquisition. The survey makes some important observations about properties common to all security metrics. These are presented as binary values which may not be suitable for all metrics, but establishes universal metric attributes we can use in any system.

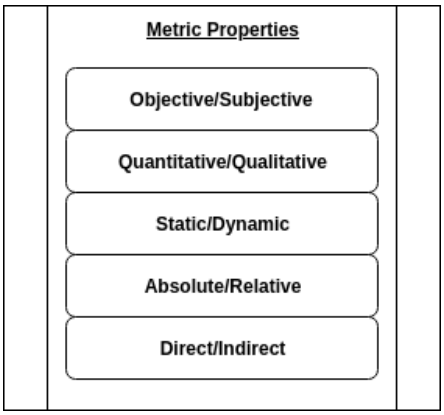


Figure 2.8: Vaughn's metric properties[411]

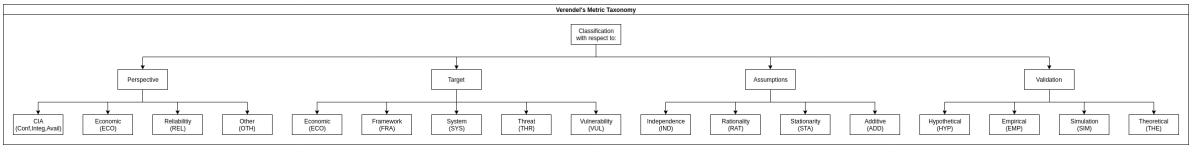


Figure 2.9: Verendel's Security Metric Taxonomy [415]

Verendel's survey[415] is a critical analysis of the claim that security is quantifiable. The premise is that most of the published models and metrics that attempt to measure security lack the scientific rigor to corroborate or validate their hypothesis. The scope of [415] is limited to operational security measurements and assumes measurement primitives include systems, threats, and vulnerabilities. 90 sources published between 1981 and 2008 were surveyed (down selected from 140). These 90 sources are then classified on 4 properties:

- **Perspective**: describes the approach taken to security. CIA, ECO, REL, OTH
- **Target**: what the source attempts to quantify.ECO, FRA, SYS, THR, VUL
- **Assumptions**: assumptions made by the source: IND, RAT, STA, ADD
- **Validation**: how the source supported findings: HYP, EMP, SIM, THE

Verendel shows that some classes of metrics, specifically cryptographic strength and intrusion detection performance, are validated frequently in the literature through commonly understood methods, while the remaining metric classes are insufficiently validated.

Pendleton's survey[308, 309] approach focuses on metrics that quantify attack and defense interactions. Metrics from 158 sources are classified as measuring one or more of Vulnerabilities, Threats, Defenses, Situations. Situations in this case is a comprehensive metric,

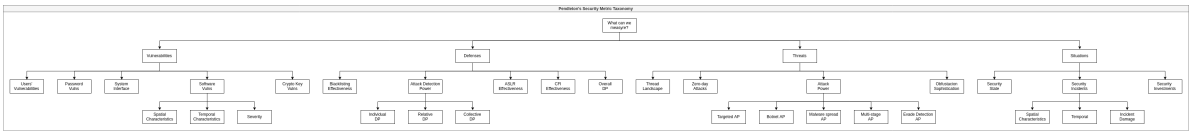


Figure 2.10: Pendleton's Security Metric Taxonomy[309]

with Pendleton's example subgroups measuring security state over time, successful attacks over time (incident rate), and return on economic investment. Select metrics aligned with Pendleton's survey are shown in Table 2.3.

Measuring what?	Representative Metrics Systemized in Paper	Desirable Security Metrics
Measuring System Vulnerabilities		
users' vulnerabilities	user's susceptibility to phishing attacks [367], user's susceptibility to malware infection [218]	user's susceptibility to class(es) of attacks (e.g., social-engineering)
password vulnerabilities	parameterized/statistical password guessability [431, 61, 201, 410]	worst-case and average-case parameterized[62]
	password entropy[71]	password guessability
system interface	attack surface [251], exercised attack-surface[283]	interface-induced susceptibility
software vulnerabilities	unpatched vulnerabilities[95], exploited vulnerabilities [283], vulnerability prevalence [459]	
vulnerability spatial characteristics		vulnerability situation awareness
vulnerability temporal characteristics	historical (exploited) vulnerability[364, 19] , future (exploited) vulnerability [364, 19], tendency-to-be-exploited [343], vulnerability life-time [142, 282, 453, 130, 459],	vulnerability vector at any time†, distribution of vulnerability lifetime
vulnerability severity	CVSS score [of Incident Response and (FIRST)], availability of exploit [52]	patching priority† , global damage
cryptographic key vulnerabilities	vulnerable cryptographic keys [453, 130] Heninger et al. 2012	(avoidable via prudential engineering)
Measuring Defenses		
effectiveness of blacklisting	reaction time [214] coverage[214]	blacklisting probability
attack detection power		
individual detection power	detection time[323], false-positive, false-negative, true-positive, true-negative, ROC, intrusion	detection probability
	detection capability[157], cost[146]	
relative detection power	relative effectiveness[56, 55]	relative effectiveness against unknown attacks
collective detection power	collective effectiveness [56, 274, 55, 272, 451]	collective effectiveness against unknown attacks
ASLR effectiveness	entropy [363], effective entropy[169]	security gain†, extra attack effort†
CFI effectiveness	CFG accuracy [Evans et al. 2015],	CFI resilience† , CFI power
overall defense power	penetration resistance[230], indirect MTD effectiveness[162]	resistance against unknown attacks†, direct MTD effectiveness

Measuring what?	Representative Metrics Systemized in Paper	Desirable Security Metrics
	Measuring Threats	
threat landscape	exploit kits [Ablon et al. 2014], malicious network[461], rogue network [389], ISP badness [193] control-plan reputation[208], early-detection time[208], cybersecurity posture[456], sweep-time[456], attackrate [456]	comprehensive cyber threat posture
zero-day attacks	number of zero-day attacks [Corporation 2012], lifetime of zero-day attacks[52], number of zero-day attack victims [52]	susceptibility of a computer to zero-day attacks
attack power		
power of targeted attacks	targeted threat index[165]	susceptibility to targeted attacks
power of botnet	botnet size[110], botnet efficiency[111], botnet robustness[111]	botnet attack power, botnet resilience with counter-countermeasures
power of malware spreading	infection rate[91]	attack power , wasted scans
power of multi-stage attacks	necessary defense[369], weakest adversary[304], attack paths [333, 369, 189, 93], k-zero-day-safety [429], effort-to-security-failure[109, 297]	multi-stage attack power
power of evading detection	(no nontrivial metrics defined)	evasion capability
obfuscation sophistication	obfuscation prevalence[340], packer structural complexity[407]	obfuscation sophistication
	Measuring Situations	
security state	fractions of compromised computers at time t, probability a computer is compromised at time t[228, 106, 445]	S(t) and si (t) for any security incidents
incident spatial characteristics	incident rate [268, 452, 218]; Maier et al., encounter rate[452, 266, 268, 218]	incident occurrence frequency
incident temporal characteristics	delay in incident detection [for Internet Security 2010], time between incidents [for Internet Security 2010; [195, 249, 172], time-to-first-compromise [195, 249, 172]	predictive incident occurrence frequency†
incident damage	cost of incidents [for Internet Security 2010]	predictive incident damage†
security investments	security spending[95], security budget [for Internet Security 2010]	payoff of security investment

Table 2.3: Pendleton's Survey: Selected Attack & Defense Metrics[309]

In the conclusions Pendleton hints at some properties desirable in all security metrics (additivity) but stops short of declaring these necessary traits for validation, or even enumerating the full list of common metric attributes.

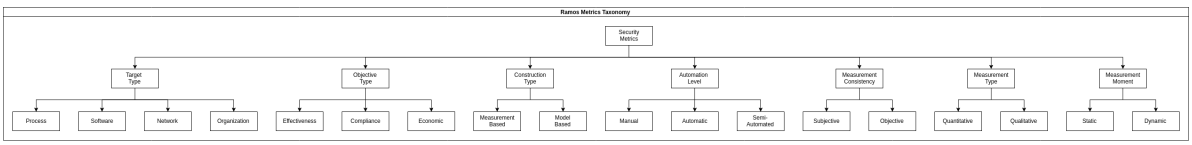


Figure 2.11: Ramos' Security Metric Taxonomy[327]

Ramos[327] focuses on model-based network security metrics exclusively, and provides a list of 5 properties distinct from Vaughn's in [411] which all good metrics should possess. Again we see validation listed as necessary to all types of metrics. Ramos cites 146 sources in the survey and consolidates classification to around 75 distinct metrics.

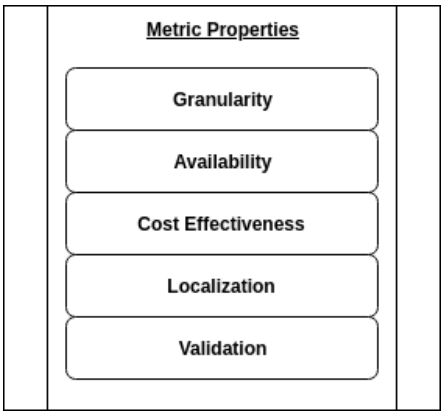


Figure 2.12: Ramos' Model Based Security Metric Properties[327]

The primary classification in [327] is by target. A metric can evaluate a Process (eg SSE-CMM), Software, the Network, or the Organization - which includes physical and personnel security metrics. The Construction Type category distinguishes between empirical and analytical metrics, the latter requiring some type of model (attack graph, markov, etc) to perform evaluation. Measurement Consistency describes whether a metric is objective or subjective.

Ramos splits the set of model based quantitative security metrics into 3 buckets based on the input model the metric expects (Stochastic, Graph, Other) with other including attack nets, petri nets, etc. This partitioning is likely to make reporting results easier as, in our experience, these input models will be produced from the same set of input data. Should we consider a metric that computes a function analytically and another that estimates the same quantity through simulation separate metrics? In table IX[327] Ramos indicates the lack of validation across the surveyed metrics even when the author's own inline validation we accounted for. Select metrics aligned with Ramos' survey are shown in Table 2.4.

Metric	Compliance	Moment	Consistency
MTTF [107][109]	compliance	dynamic	objective
METF [297]	compliance	dynamic	subjective
MTSF [27], [26], [25]	compliance	static	subjective
MTFF [217], [347]	compliance	static	subjective
MTTC by McQueen et al. [256]	compliance	static	subjective
MTTC by Leversage et al. [229]	compliance	static	subjective
Steady-State Security [27], [26], [25]	non-compliance	static	subjective
Reliability [189]	compliance	static	objective
Success Likelihood [199]	non-compliance	dynamic	subjective
q [232]	non-compliance	static	objective
Shortest Path [315], [182]	compliance	dynamic	objective
Number of Paths [297], [182]	non-compliance	dynamic	objective
Mean of Path Lengths [231], [182]	compliance	dynamic	objective
Normalized Mean of Path Lengths[182]	compliance	dynamic	objective
Assistive metrics: SDPL, MoPL, MePL [182]	compliance	dynamic	objective
Weakest Adversary [304]	compliance	static	subjective
Network Compromise Percentage [235]	non-compliance	dynamic	objective
Network Compromise Percentage [235]	non-compliance	dynamic	objective
State Rank [258]	non-compliance	static	subjective
Cumulative Score [291]	non-compliance	static	objective
AGP [427]	non-compliance	static	subjective
Attack Resistance [425]	compliance	static	subjective
Enhanced Cumulative Score [174]	non-compliance	static	subjective
Liu and Man's metric [241]	non-compliance	dynamic	subjective
Frigault and Wang's metric [144]	non-compliance	static	subjective
Frigault and colleagues' metric [143]	non-compliance	dynamic	subjective
Poolsappasit and colleagues' metric [318]	non-compliance	dynamic	subjective
Xie and colleagues' metric [438]	non-compliance	dynamic	subjective
Dantu and colleagues' metric [115], [114], [116]	non-compliance	dynamic	subjective
Expected Difficulty [148]	compliance	static	subjective
VEA-bility [406]	compliance	static	subjective
k-zero day safety [428], [429]	compliance	static	subjective
d2-Diversity (least attacking effort) [460], [423]	compliance	static	subjective
d3-Diversity (avg. attacking effort) [460], [423]	non-compliance	static	subjective
d1-Diversity (% of distinct resources) [460], [423]	compliance	static	subjective
Seclius [467]	non-compliance	dynamic	objective
Damage risk [88]	non-compliance	static	subjective
Mean Privacy [27]	non-compliance	static	subjective
Security Meter [345], [344]	non-compliance	static	subjective
Policy Security Score [8]	compliance	dynamic	subjective
Probabilistic Vulnerability Measure [19]	non-compliance	dynamic	subjective
Attack Propagation [20]	non-compliance	dynamic	subjective

Metric	Compliance	Moment	Consistency
ADVISE [228]	compliance	static	subjective

Table 2.4: Ramos' Survey: Selected Model Based Security Metrics[327]

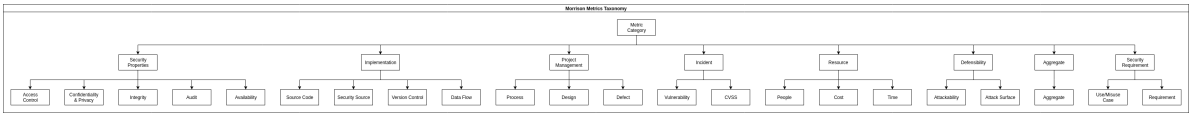


Figure 2.13: Morrison's Security Metric Taxonomy[277]

Morrison surveys 71 sources (down selected from 4818) to classify 324 security metrics from the SDLC. The number of metrics for each group are summarized in table 2.14.

Metric Name	*Category*	Paper	Method	Scale	Phase
Mechanism strength	Design	[241]	Quantitative	count	Implementation
Action Register	Access Control	[417]	Qualitative	count	Operations
Alphabet Size	Access Control	[417]	Qualitative	Enumeration	Operations
Authentication Period	Access Control	[417]	Qualitative	Enumeration	Operations
Block by User Cancellation	Access Control	[417]	Qualitative	ordinal	Operations
Group Password	Access Control	[417]	Quantitative	count	Operations
Information about Use	Access Control	[417]	Qualitative	Enumeration	Operations
Initial Communication	Access Control	[417]	Quantitative	duration	Operations
Input Visualization	Access Control	[417]	Qualitative	Enumeration	Operations
Maximum Life Time	Access Control	[417]		Not specified	Implementation
Maximum Number of Erroneous Attempts	Access Control	[417]	Qualitative	ordinal	Operations
Minimum Length	Access Control	[417]	Qualitative	ordinal	Operations
Minimum Life Time	Access Control	[417]		Not specified	Operations
Net Transmission	Access Control	[417]	Qualitative	Enumeration	Operations
Number of Different Classes	Access Control	[417]		Not specified	Operations
Password Reassigning	Access Control	[417]	Quantitative	Time	Operations
Predefined Users	Access Control	[417]	Qualitative	ordinal	Operations
Record Length	Access Control	[417]	Qualitative	ordinal	Operations
Selection Restriction	Access Control	[417]	Qualitative	ordinal	Operations
Source Selection	Access Control	[417]	Qualitative	ordinal	Operations
Storage Class	Confidentiality and Privacy	[417]	Qualitative	Enumeration	Operations
User Identificator	Access Control	[417]		Not specified	Operations
User Training	Access Control	[417]	Quantitative	duration	Operations
CVSS Score	CVSS	[359]	Quantitative	count	Operations
Patch Index	Vulnerability	[359]	Quantitative	count	Operations
Annual Loss Expectancy (ALE)	Cost	[58]	Quantitative	duration	Operations

Metric Name	*Category*	**Paper**	**Method**	**Scale**	**Phase**
Return on Penetration Testing	Cost	[58]	Qualitative	ordinal	Operations
Business Adjusted Risk	Cost	[403]	Quantitative	count	Operations
Daily Vulnerability Exposure	Vulnerability	[403]	Qualitative	ordinal	Operations
Vulnerability Index	Vulnerability	[403]	Qualitative	ordinal	Operations
CN Betweenness	People	[264]	Quantitative	count	Operations
DN Max Edge Betweenness	People	[264]	Quantitative	Time	Operations
Num Commits	Version Control	[264]	Quantitative	Not specified	Operations
NumDevs	People	[264]	Qualitative	ordinal	Operations
Vulnerability	Vulnerability	[264]	Qualitative	ordinal	Operations
Contributions	Version Control	[310]	Qualitative	ordinal	Operations
Fork count	Version Control	[310]		Not specified	Operations
Number of commits	Version Control	[310]	Qualitative	ordinal	Operations
Number of hunks	Version Control	[310]	Qualitative	ordinal	Operations
Patch	Version Control	[310]	Quantitative		Operations
Patch keywords	Version Control	[310]	Quantitative	count	Operations
Programming language	Source Code	[310]	Qualitative	currency	Operations
Star count	Version Control	[310]	Quantitative	ratio	Operations
Vulnerability	Vulnerability	[310]	Qualitative	currency	Testing
Probability Compromised and Repaired (PC)	Attackability	[252]		count	Operations
Probability Compromised Not Repaired (PCNR)	Attackability	[252]	Quantitative	Classes	Design
Probability Secure (PS)	Attackability	[252]	Quantitative	probability	All
Probability Unpatched Compromise (PPC)	Attackability	[252]	Quantitative	ratio	Design
Vulnerability Disclosure	Vulnerability	[252]		probability	Operations
Vulnerability Discovery	Vulnerability	[252]		count	Operations
Vulnerability Patch	Vulnerability	[252]		count	Operations
Access Complexity (AC)	CVSS	[354]		count	Operations
Access Vector (AV)	CVSS	[354]	Quantitative	Classes	Design
Authentication (AU)	CVSS	[354]		probability	Operations
Availability Impact (A)	CVSS	[354]	Quantitative	Ratio	Requirements
Availability Requirement (AR)	CVSS	[354]	Quantitative	probability	Operations
Collateral Damage Potential (CDP)	CVSS	[354]	Quantitative	probability	Operations
Confidentiality Impact (C)	CVSS	[354]	Quantitative	probability	Operations
Confidentiality Requirement (CR)	CVSS	[354]	Quantitative	probability	Operations
Exploitability (TE)	CVSS	[354]	Quantitative	count	Design
Integrity Impact (I)	CVSS	[354]		count	Operations
Integrity Requirement (IR)	CVSS	[354]	Quantitative	count	Operations
Remediation Level (RL)	CVSS	[354]	Quantitative	count	Operations
Report Confidence (RC)	CVSS	[354]	Quantitative	count	Operations

Table 2.5: Morrison's Survey: Selected Software Security Metrics[277]

Table 4
Security metric categories and dubcategories.

Metric category	Subcategory	Subcat Total	Cat Total
Security Properties			86
	Access Control	42	
	Confidentiality and Privacy	19	
	Integrity	10	
	Audit	9	
	Availability	6	
Implementation			84
	Source Code	44	
	Security Source-Code	30	
	Version Control	8	
	Data Flow	2	
Project Management			41
	Process	17	
	Design	17	
	Defect	7	
Incident		36	
	Vulnerability	20	
	CVSS	16	
Resource			34
	People	23	
	Cost	9	
	Time	2	
Defensibility			19
	Attackability	10	
	Attack Surface	9	
Aggregate			18
	Aggregate	18	
Security Requirement			16
	Usecase-Misusecase	12	
	Requirement	4	
Grand Total			324

Figure 2.14: # Security Metrics by Category/SubCategory (out of 324)[277]

As these metrics are focused on software security, Morrison finds that many are either normalized to existing software metrics or are extensions thereof. Subcategories of Security Properties seem to be based on surveys about the listed property but this may reflect the source authors rather than an inability to automate collection. Incident Metrics more closely map to previous other's Vulnerabilities categories as nothing indicates successful attacks are being examined here. From the paper's findings, 85% of metrics surveyed have only been proposed and evaluated by the author, pointing again to the need for metric validation. Very few metrics apply to design time or test time evaluation in the SDLC - most are tuned for production deployment. The majority of metrics are subjective, relying on user feedback. Select metrics aligned with Morrison's survey are shown in Table 2.5.

2.2.2. A Unified Security Metrics Taxonomy

Above we reviewed some existing classification methods for security metrics. These included taxonomies from both narrow and broadly focused surveys. While there is certainly overlap within some of the categories and properties identified, there is also collision between similar sounding concepts from different sources. To remove ambiguity, and more importantly to address the immediate question, we can map all our security metrics onto the taxonomy derived from the Cyber Security Body of Knowledge. The CyBoK is an effort to collect and maintain canonical research across the entirety of the cyber security domain. Included in this mandate is keeping all of it organized, so we can presume that if there is a topic in security we would like to measure, then there will be a corresponding topic in the CyBoK to consult.

Security metrics can be categorized by the area of cyber security to which they apply. In this respect, the security metric surveys available in recent literature are by nature focused narrowly on a specific subfield, such as cryptographic or software development lifecycle security metrics. To remove ambiguity in terms among surveys, we attempt to include these in a *big-picture* view of the field of cyber security by classifying them under the general headings of the recently released Cyber Security Body of Knowledge[329] depicted in Figure 1.1. By grouping our security metrics by Cybok category, we can determine our cyber security metric coverage, and use this context to identify non-security related metrics that would be relevant to the area.

2.2.2.1. Human, Organization, & Regulatory Metrics

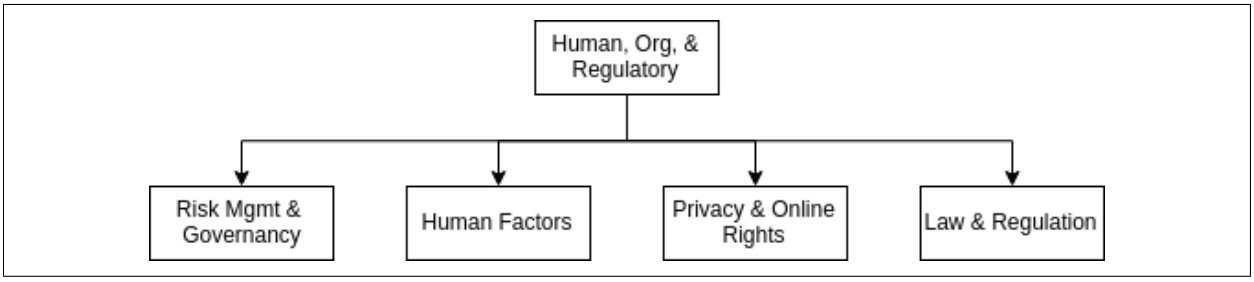


Figure 2.15: Cybok: Human, Organization, & Regulation Metrics

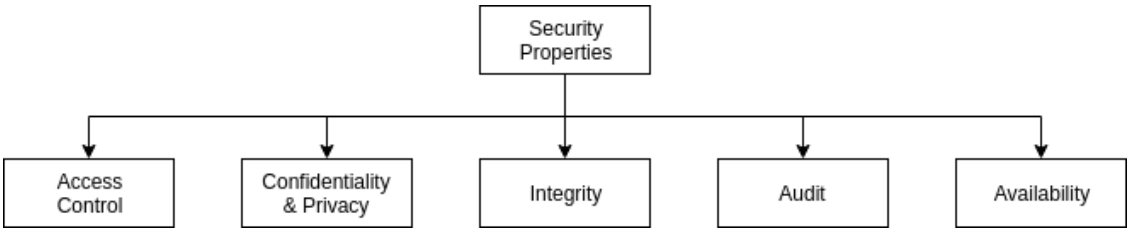


Figure 2.16: Most of Morrison's class for Security Properties would fit under HO&R.

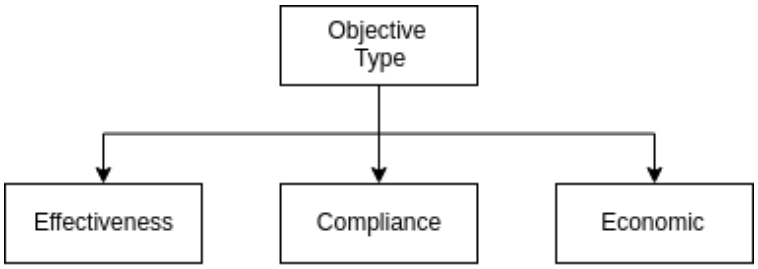


Figure 2.17: Ramos' Objective Type metrics fall under HO&R (economics is cross cutting).

Metrics in HO&R are usually derived from applicable regulations and policies (ISO, FISMA, HIPPA, PCI, ADA, etc). Often these are counts or ratios that quantify the proportion of assets that are in/out of compliance with the regulation. Typical applications for these metrics are system audits (how many current users have completed mandatory training) or accreditations (how many of these secure operations checkboxes does the current system check).

2.2.2.2. Attack & Defense Metrics

Attack and defense describes many of the security metrics we have investigated in this thesis. Malware and Attack metrics can quantify an attacker's capabilities, the DoS'ing bandwidth of a botnet or the number of accounts controlled are examples. Adversarial behaviours might relate to MITRE's APT and CAPEC attack pattern datasets, but I haven't encountered metrics that evaluate this yet (although it shows up regularly in threat models). Forensics metrics typically include time to unpack or deobfuscate a malware sample, or the amount of time to determine an indicator of compromise for IDS deployment. SecOps & Incident Response metrics include standards from the literature such as IDS efficacy and

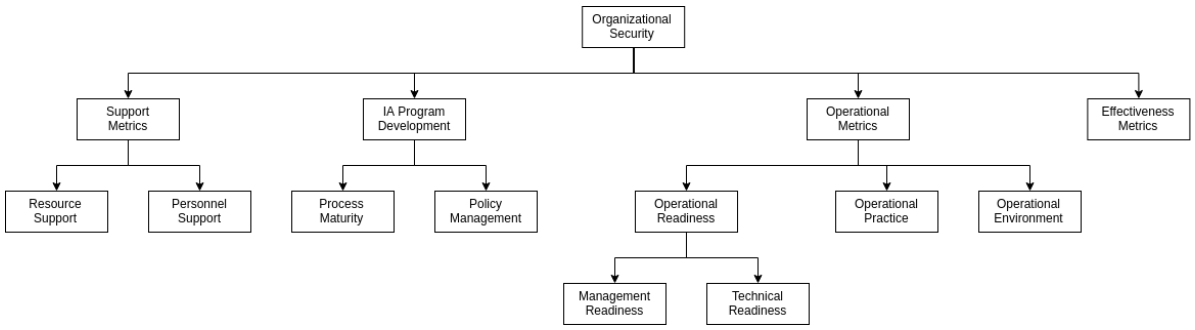


Figure 2.18: Vaughn's Organization Security metrics subtree falls under HO&R.

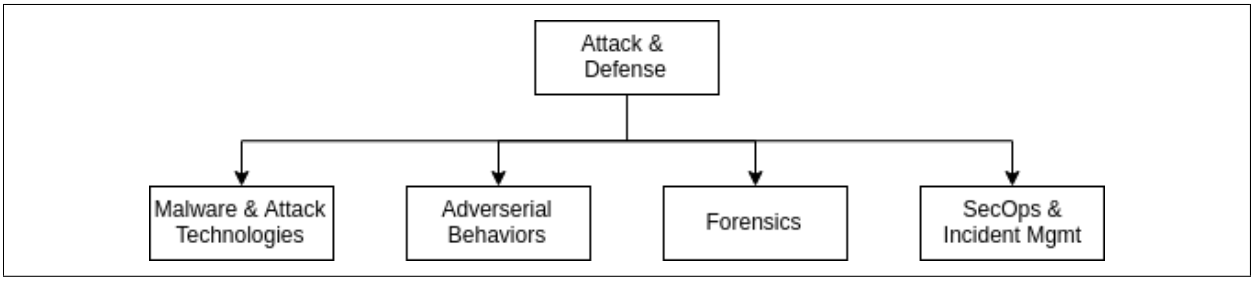


Figure 2.19: Cybok: Attack & Defense Domain Metrics

mean time between failures.

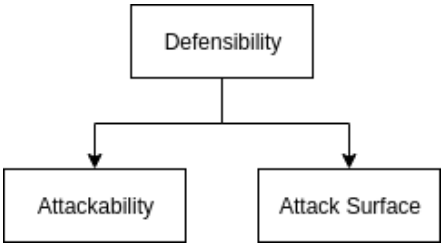


Figure 2.20: Morrison's Defensibility metrics fall generally under A&D.

2.2.2.3. Systems Security Metrics

Systems security metrics span most aspects of operational security. Hardware and Network security metrics were discussed under infrastructure. Typical applications of cryptographic security metrics include all the formal verification artifacts involved in the validation of a protocol or implementation, along with performance, key size, entropy, etc. OS security metrics may be derived from common criteria/ EAL or measure isolation, weakness to side channels, or number of vulnerabilities known along with severity.

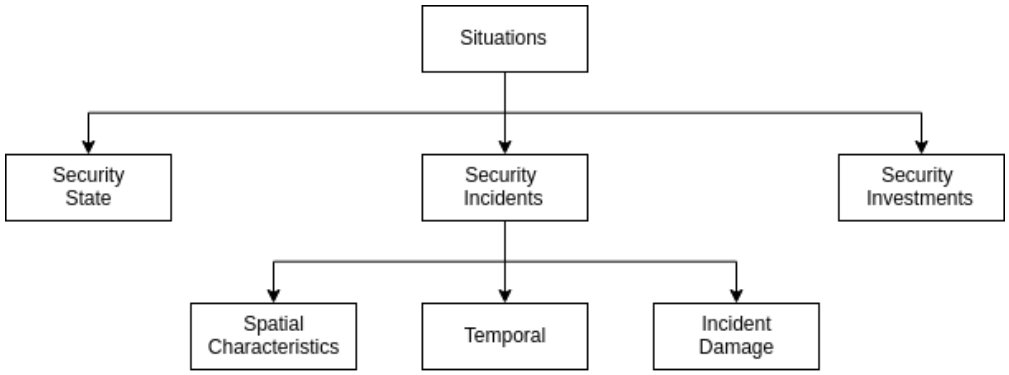


Figure 2.21: Pendleton's Situations metrics fall under A&D.

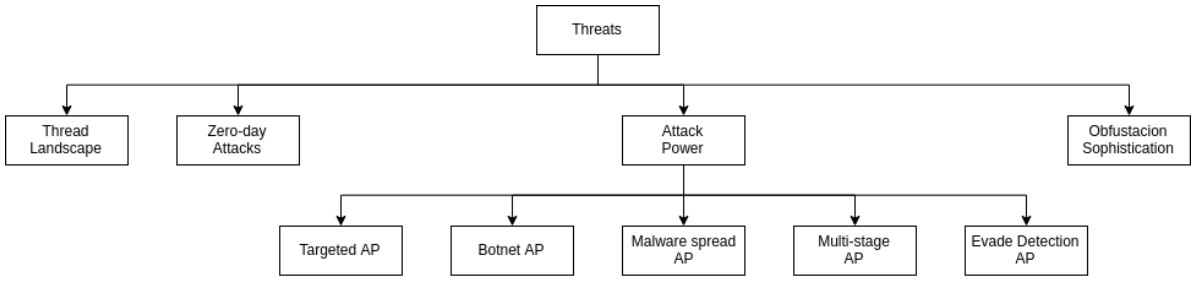


Figure 2.22: Pendleton's Threats metrics fall under A&D.

Model based network security metrics classified by Ramos[327] are shown in Figure 2.11. These metrics are designated as Compliance based when a larger value indicates more security. Moment identifies if a measurement can be taken pre-deployment and remains static throughout operation, or if the metric is dynamic and should be measured repeatedly. Consistency distinguishes if the measured value relies on subjective human input or if its evaluation is objective. To compare with network performance metrics like latency or throughput, these security metrics are heavily influenced by subjective criteria. For example, the reliability based models make assumptions about an attacker's success rate, level of effort, motivations, and capabilities that could change depending on who is filling in the weights.

2.2.2.4. Infrastructure Security Metrics

Infrastructure Metrics to evaluate hardware security can be found in Rostami's survey[338, 339]. The majority of these are incident counts or ratios of expected to actual values, although analytical calculations (Hamming distances) are suggested to measure the

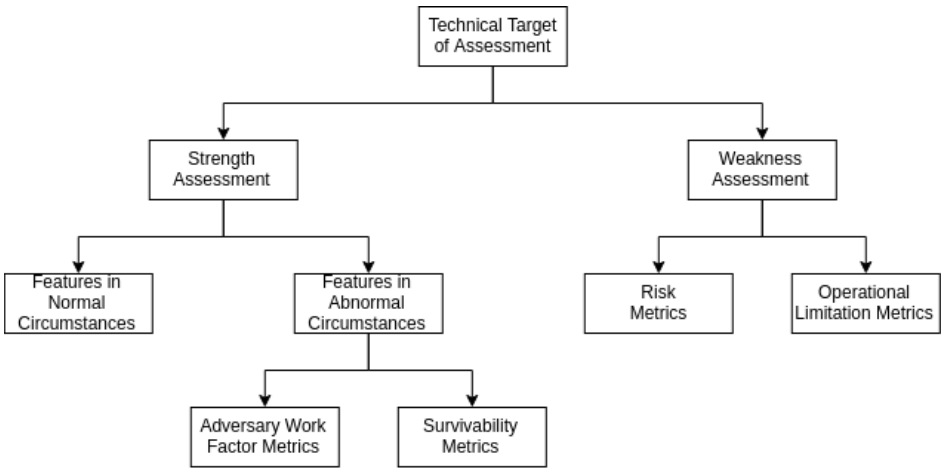


Figure 2.23: Vaughn's Strength and Weakness metrics fall under A&D.

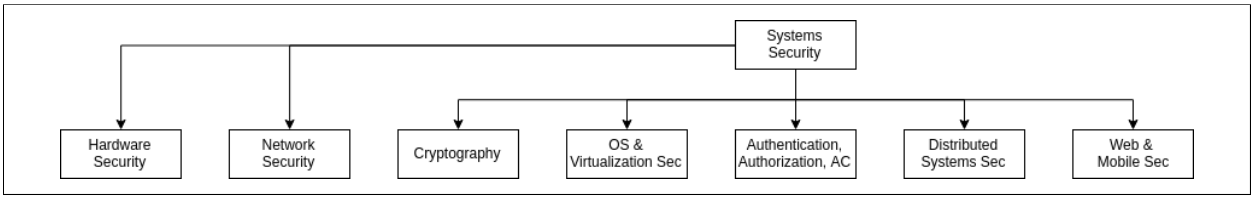


Figure 2.24: Cybok: Systems Domain Metrics

amount of divergence from a known good source in several proposed metrics. Assuming the gold standard is available against which these measurements can be taken, they will evaluate the measurement empirically and deterministically. These metrics are similar to their performance related counterparts (eg, SPEC CPU) in that the performance increase over or under a reference system can be represented as a ratio of the two values.

Cyber-Physical Systems (CPS) includes SCADA and other control systems, vehicle networks, and IoT systems that fall outside the scope of this thesis but certainly have applicable security metrics associated with attack surface and information leakage. Similarly, most aspects of the other infrastructure components listed above are captured in the system and threat model and included in the Attack & Defense security metrics. Types of security metrics applicable here but not listed in the surveys above might include supply chain vulnerabilities, weakness to eavesdropping or side channel attacks.

2.2.2.5. Software & Platform Security Metrics

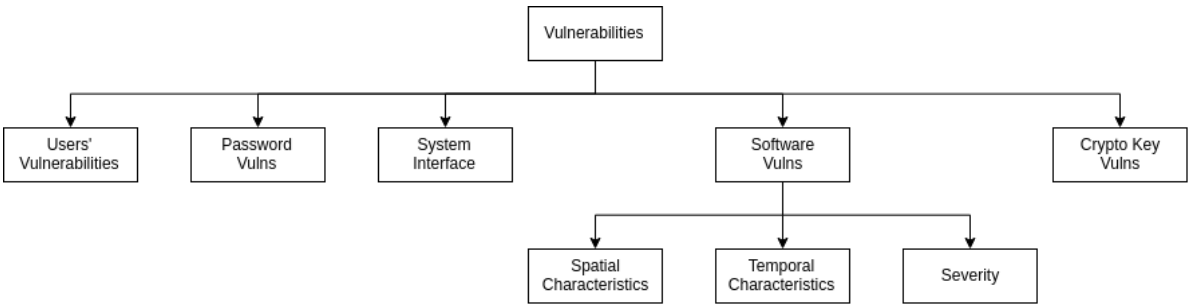


Figure 2.25: Pendleton's Vulnerabilities security metrics fall under System's security metrics

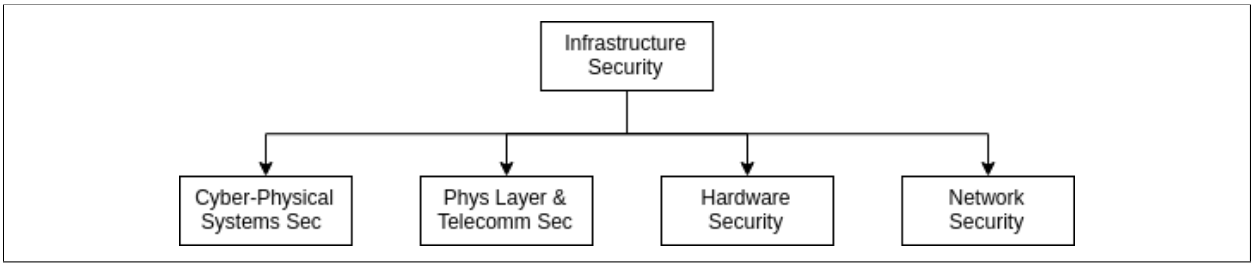


Figure 2.26: Cybok: Infrastructure Domain Metrics

Software security metrics were the focus of Morrison's survey and apply here broadly. Typical applications of these type metrics derive from static analysis and test coverage. Of specific interest in the thesis is the remediation velocity, which measures the time between discovering a flaw in software and the time a fix has been merged into the code base.

2.2.3. Summary

In the surveys summarized above there were over 500 distinct security metrics identified. The surveys each provided their own classification systems which were appropriate for the analysis they conducted, but none of these taxonomies generalize well to classify all types of security metrics. In this section we have described properties common to all metrics, identified overlaps in the various taxonomies, identified points of confusion between existing metric hierarchies, and described a suitable and intuitive system for classifying any current or future security metric. By using the Cybok as the underlying classification system we are also able to determine the distribution of metrics in each topic and identify areas of limited coverage which would benefit from future research.

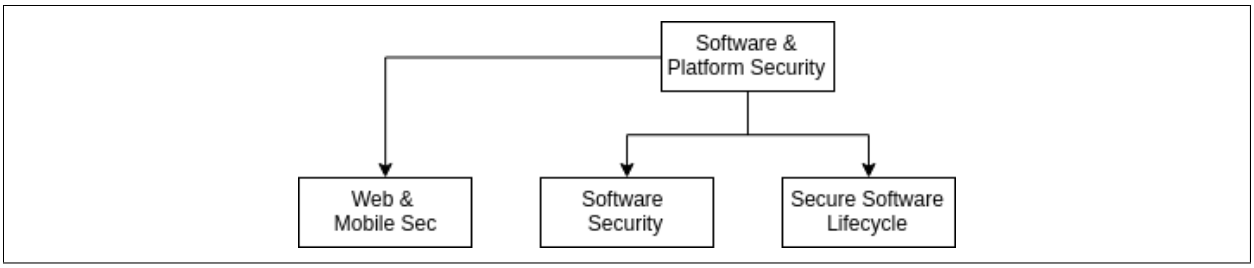


Figure 2.27: Cybok: Software & Platforms Domain Metrics

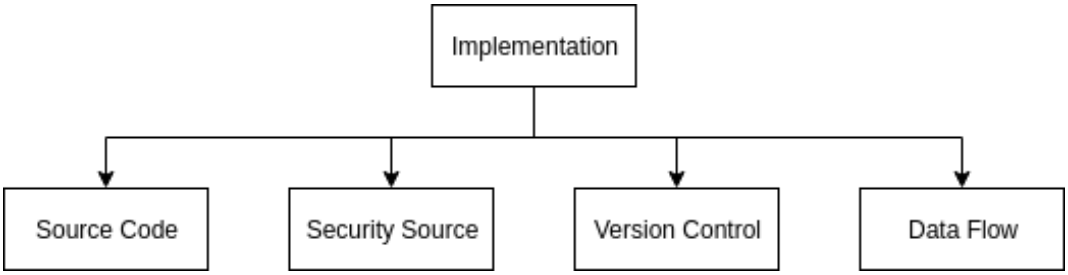


Figure 2.28: Morrison's Implementation Security metrics fall under SW&P.

2.3. Analytics Pipelines

So far we have reviewed an assortment of models, which describe the syntax and grammar of the system components we wish to analyze, along with metrics, which capture some property of security we wish to quantify. To measure a specific security property a metric is applied to an appropriate collection of model instances. This mapping is usually implicit in the metric's definition - a vulnerability based metric assumes a vulnerability scanner has been run against a system and the results are packed in an appropriate format for the metric to operate on for example. Obtaining a measurement from input model instances may be trivial or could involve multiple pre-processing steps. We can think of the steps taken for a security measurement to be obtained as a processing pipeline for security analytics, and approach these steps like an extract, transform, load (ETL) pipeline in Chapter 3. For background we present in this section the analytics pipeline proposed in [9], as it forms the basis of the case-study in Section 6.1, and motivates much of the developed automation used in future chapters of this thesis.

2.3.1. Cyber Security Analytics Framework

The Cyber Security Analytics Framework[9] is a modular process for evaluating system security. CSAF provides a suite of metrics designed to measure facets of system integrity, providing a means of quantitative comparison that directly lends itself to system migration planning. The process follows the basic *input-process-output* sequence established in the *Multi-host, Multi-stage Vulnerability Analysis Language*[299] toolkit MulVal.

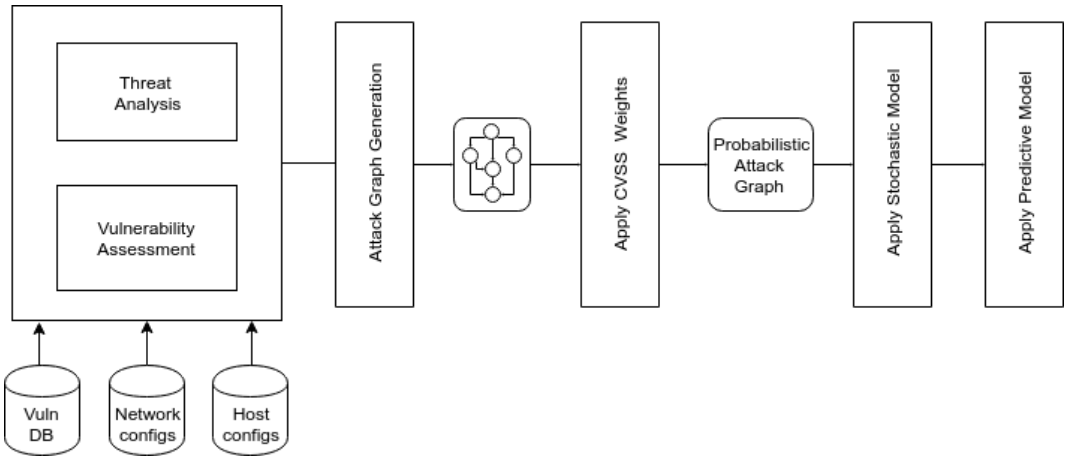


Figure 2.29: Cyber Security Analytics Framework[9]

MulVal provides an efficient[328] Datalog based modeling language and inference engine for vulnerability relationship analysis. The input model consists of configurations for **hosts** and **networks**, **principals** describing user accounts and privileges, rules governing **interactions**, and access **policies**. Information about the system under test is either gathered through standard methods[301] such as Nessus, Nmap, and OVAL scans or defined manually for hypothetical data points, and parsed into the global system model using an appropriate adaptor. MulVal can then make inferences about how the entities are able to interact. When provided with a starting and ending node, MulVal enumerates all possible paths an attacker may traverse to compromise the target. The attack graph represents the exploitable vulnerabilities in a system as the set of connected nodes and edges between an attacker's origin and the target.

Once the attack graph has been generated, the processing stages of the CSAF collect known vulnerabilities, apply weights to the enumerated attack paths, run simulations, and

compute analytical measurements for the set of security metrics desired. These measurements are reported for each state model to provide a clear view of system security along a migration path, or to support comparison between competing target systems. The remainder of this section will describe these steps in more detail.

2.3.2. CSAF Metrics Foundation

Table 2.6: Metrics Summary

Metric Class	Description	Common Measurements
Structural	Metrics based on the structure of the attack graph; used to identify attributes like shortest path, mean path length, or total number of paths.	SP, NP, MPL
Time-Based	Metrics that quantify time expectations for attributes like compromise, recovery, or incident response.	MTTF, MTTB, MTTR
Probability-Based	Metrics that associate probabilities attack paths to quantify the security of the network.	NR, PP, EPL
Temporal	Metrics that examine vulnerability age on the system.	TAG

Structural metrics draw conclusions about the security properties of the attack graph through basic graph analysis techniques[109][297].

Shortest Path (SP):

Given an attack graph, the Shortest Path metric identifies the minimum number of nodes (vulnerabilities) an attacker would need to exploit to reach the target. Techniques for finding the shortest path in a graph are well-documented in Computer Science [124]. For the collection of paths, p_i, in an attack graph AG we define the shortest path as:

$$SP(AG) = min(len(p1), len(p2), \ldots, len(pi), \ldots, len(pn))$$

Number of Paths (NP):

NP is a count of the unique paths that exist on an attack graph between the attacker and the target. It is a reasonable measure of the risk exposure of the network and provides a sense of how many options an attacker would have available during a targeted attack.

$$NP = |p_1, p_2, \ldots, p_i, \ldots, p_n|$$

Mean Path Length (MPL):

MPL calculates the arithmetic mean of the path lengths on the network as a way to size the average effort required to compromise a target.

$$MPL = \frac{\sum_i len(p_i)}{NP(AG)}$$

While we can obtain some insight into the security properties of different attack graphs through direct comparison, there is not enough granularity in these structural metrics to determine the characteristic strengths or weaknesses of the underlying security posture. For example, we notice that there is a large discrepancy in the NPL measures among the three graphs; however, we can't determine conclusively that this makes one model more susceptible to attack without knowing more about how each model's vulnerability and exploitability relate. Effort has been made [12] to introduce statistical methods into structural metrics as a means for more reliable comparison.

Attack graphs have long been modeled as probabilistic processes[109, 297, 315, 432]. We consider the movement of an attacker through the nodes of the attack graph as a stochastic process and interpret the state of the system as the current position of the attacker in our network. An attacker can advance to another state in the process through successful compromise of the vulnerability represented by that state only if there exists an edge in the attack graph between the attacker's current state and the advance state. The collection of all states in the process is the system's state space and corresponds to the set of nodes in our attack graph. Advancing to another state is probabilistic and the success of the advance is based on the weighted score associated with that node. For example, from Figure 3.9a, if

the attacker is at Node 3, the probability that the target will be compromised is only based on the difficulty of exploiting the vulnerability on Node 1 (Trojan installation), and not on the path the attacker took to arrive at Node 3.

Because we only need to rely on the current state of the system and not how the system arrived in that state to determine the next state, we are able to model the attack graph as a Markov Chain without loss of generality. A Markov Chain is a stochastic process that is *stateless*, that is, prediction of the next system state can be made based only on the current state. This is known as the Markov Property.

More formally, for a stochastic process $X = X_t, t \geq 0$, if the attack graph has n nodes, then the set of possible states, S, for X is $S = s_1, s_2, \ldots, s_a, \ldots, s_n$. The probability P_i, j that an attacker in state X_t will advance to state X_{t+1} can be given as $P(X_{t+1} = j | X_t = i)$ with the Markov Property being satisfied as:

$$P(X_{t+1} = j | X_1 = x_1, X_2 = x_2, \ldots, X_t = i) = P(X_{t+1} = j | X_t = i)$$

The value $P_{i,j}$ is known as the transition probability between two statess_i and s_j. We can model the n states of the process X as an nxn matrix whose(i, j)entry is the transition probability$P_{i,j}$.

The transition matrix

$$P = \begin{bmatrix} P_{11} & \ldots & P_{1n} \\ \vdots & \ddots & \\ P_{n1} & & P_{nn} \end{bmatrix}$$

must also satisfy the conditions:

$$0 \leq P_{i,j} \leq 1 \quad for \quad i, j \in S$$

$$\sum_{j=1}^{n} P_{i,j} = 1$$

That is, if a state has three outbound edges (possible choices to exploit next), the probability that any edge is followed is 1 since we must proceed to the attack goal after each time step, and the probability a specific edge is followed is determined by how exploitable that vulnerability is (the transition probability). This stochastic model enables us to study the system's quantitative and qualitative properties through well-established analytic and simulation methods. Assuming the system state is given as the attacker's current location on the attack graph (the vulnerability most recently exploited), we can model the system's subsequent states through iteration of the stochastic process over discrete time intervals.

Attack graphs in general have the special property that the attack goal can be reached from any node in the network, allowing us to model them as an Absorbing Markov Chain using the transition matrix described above. An Absorbing Markov Chain is a Markov Chain that includes at least one absorbing state, in our case the attack goal. The absorbing state can be reached from all other states, and once the absorbing state is reached (the attacker has compromised the target), no further transitions are considered.

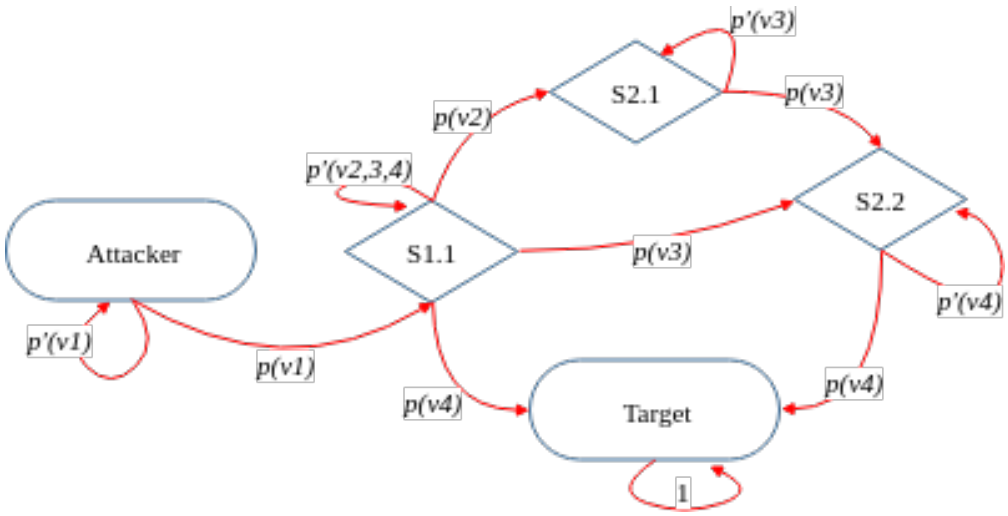


Figure 2.30: Example Transition Diagram

In Figure 2.30 we see a notional transition diagram. The possible states in the chain are connected by edges labeled with the probability of advancing to an adjacent state (successfully exploiting the next vulnerability.) The self-referencing edges represent the probability of an unsuccessful exploit and occur at entries Pi, i along the diagonal of the transition ma-

trix. Note that once the system enters the 'Target' state no other state is reachable which we define as the absorbing state.

To create a transition matrix that conforms to the definition of an absorbing Markov chain, each outbound edge is assigned a transition probability calculated by normalizing the CVSS exploitability scores associated with all adjacent (next-step) states. That is, if we are at state $S_i$and the set of possible next states are given as $S_{i+1} = s1, s2, \ldots, sn$ then we calculate the transition probability

$$Pij = \frac{CVSS(s_j)}{\sum_{k=1}^{n} CVSS(s_k)}; k \in S_{i+1}$$

This normalizes the transition probabilities for all outbound states of a given node and guarantees the two conditions for a Markov transition matrix defined above will be satisfied.

The transition matrix P for the absorbing Markov chain defined above can be put into the canonical form $P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix}$ where Q is the matrix of transition probabilities for moving from a non-absorbing (transient) state to another transient state, and R is the matrix of transition probabilities for moving from a transient state to an absorbing state. In other words, we can order the rows and columns such that all transient states precede the absorbing states.

From the canonical form, P_k approaches some limiting matrix $|P|$ as k increases, where$|P|= \begin{bmatrix} 0 & FR \\ 0 & I \end{bmatrix}$ and $F = (I - Q) - 1$. This matrix F is known as the fundamental matrix for P, and it allows us to derive many interesting properties from our system. For example, the(i, j) entries of $|P|$ provide the long-term (limiting) probabilities of advancing from state i to state j. Likewise, the sum of the row entries in F determine the average number of steps it will take to reach an absorbing state from each transient state.

Note that for the transition matrix P defined above, the entry $P_{i,j}$ is the probability the system given initial state s_i will move to state s_j on the next step. It follows from the total probability theorem that the probability the system will be in state s_j after exactly two time

steps is the (i, j) entry of:

$$P^2 = \begin{bmatrix} Q & R \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Q & R \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Q^2 & QR + R \\ 0 & 1 \end{bmatrix}$$

After 3 time steps:

$$P^3 = \begin{bmatrix} Q^2 & QR + R \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Q & R \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Q^3 & Q^2R + QR + R \\ 0 & 1 \end{bmatrix}$$

. . . , and after t time steps:

$$P^t = \begin{bmatrix} Q^t & (I + Q + Q^2 ... + Q^{t-1})R \\ 0 & 1 \end{bmatrix}$$

We have defined Q as the matrix of transient probabilities with values less than 1, so it holds that $Q^t \to 0$ as $t \to \infty$ (eventually we will reach the attack target). Now consider the matrix F such that $F = I + Q + Q^2 + ...$

The values of $Q^t(i, j)$ then are the probabilities of the system starting in state s_i and ending in state s_j exactly t steps later. In this case we can interpret the probability $Q^t(i, j)$ as the fraction of time period t spent in state s_j, so the total number of periods the process occupies state s_j before becoming absorbed is: $F(i, j) = Q^0(i, j) + Q^1(i, j) + Q^2(i, j)...$ and the reduced form is known as the **fundamental matrix:**

$$F = (I - Q) - 1$$

Node Ranking (NR):

We have shown that the elements of the fundamental matrix F take on values that represent the relative duration of time spent at each transient node in the Markov process.

In the context of our security analysis, these values equate to the amount of hold time we expect an attacker to incur while trying to advance to the target. Lower node rankings indicate nodes along the attack path that are relatively easy for an attacker to clear. If a difficult to exploit vulnerability exists and its associated NR is relatively low this might be an indication that a security control point is being bypassed. Using the attack graph and associated NR analysis it is a fairly straight forward process to identify the area of interest and trace back to the origin of the bypass. Liu[239] examines this process of forensic reconstruction of attacks using attack graphs in detail.

Probabilistic Path (PP):

The PP metric is another interesting property derived from our Markov transition matrix. Taking the product of the fundamental matrix F and the matrix of absorbing probabilities R results in a matrix, $B = FR$, whose $B(i, j)$ entries yield the probability of being absorbed by state s_j given we started at initial state s_i.

Expected Path Length (EPL):

We define EPL as the expected number of time steps required for an attacker to advance from the initial state to the attack goal, and its calculation follows as a direct consequence of deriving the NR metric. That is, if the NR metric expresses the total expected time that a process starting in initial state s_i will occur in transient state s_j before ultimately being absorbed, then the NR sum over all transient states for s_i will predict the total time spent in the process before absorption. To take the sum of the values in the rows of the fundamental matrix we multiply by a column of 1's, $t = N1$, and the entry t_i contains the EPL value for initial state s_i.

Mean Time To Failure (MTTF) Time based metrics are a subset of probabilistic measures used to estimate how long it will take for some target objective to be met[109][297][256].

MTTF is the measure of the mean time for an attacker to reach a target. This measure directly maps to Expected Path Length as we have defined it.

Mean Time To Failure (MTTF)

MTTF is the measure of the mean time for an attacker to reach a target. This measure

directly maps to Expected Path Length as we have defined it above.

Mean Time To Breach (MTTB)

MTTB is the measure of the mean time for an attacker to reach a target.

Mean Time To Failure (MTTR)

MTTR is the measure of the mean time for an attacker to reach a target.

2.3.3. Summary

The Cyber Security Analytics Framework illustrates a specific example of how an analytics pipeline can be formed from the composition of existing and newly created metrics. In Chapter 3 we generalize this process as a 4-stage pipeline and show how this can be used to evaluate arbitrary sets of metrics against each other or against different inputs for validation, similarity scoring, and benchmarking. In this section we have explored the theoretical foundations of many of the model based metrics described Section 2.2 by defining

Chapter 3

Automation Driven Security Measurement

Observation is the foundation of scientific experimentation. We consider observations to be measurements when they are quantified with respect to an agreed upon scale, or measurement unit. A number of metrics have been proposed in the literature which attempt to quantify some property of cyber security, but no systematic validation has been conducted to characterize the behaviour of these metrics as measurement instruments, or to understand how the quantity being measured is related to the security of the system under test. In Chapter 2 we broadly classified the body of available security metrics against the recently released Cyber Security Body of Knowledge, and identified common attributes across metric classes which may be useful anchors for comparison. In this chapter we propose a general four stage evaluation pipeline to encapsulate the processing specifics of each metric, encouraging a separation of the actual measurement logic from the model it is often paired with in publication. Decoupling these stages allows us to systematically apply a range of input models to a set of metrics, and we demonstrate some important results in our proof of concept. First, we determine a metric's suitability for use as a measurement instrument against validation criteria like operational range, sensitivity, and precision by observing performance over controlled variations of a reference input. Then we show how evaluating multiple metrics against common reference sets allows direct comparison of results and identification of patterns in measurement performance. Consequently, development and operations teams can also use this strategy to evaluate security tradeoffs between competing input designs (which we show in the case study in Section 6.1) or to measure the effects of incremental changes during production deployments (which we show in Chapter 4).

The motivation of this thesis is to make modern information systems more secure, and the driving force behind that goal is automation. Many of the problems addressed in this

work stem from the disparate ecosystem of tools, APIs, methodologies, libraries, and frameworks that exist in relative isolation to one another. Consider Security Information and Event Management (SIEM) systems as an example, which provide correlation of host/network event logs, IDS/IPS alerts, threat/vulnerability feeds, etc, and present a unified view of the system's security posture automatically to the SOC. Before the advent of managed SIEMs, sys admins typically filled the role of security engineers, and relied on hand rolled collections of shell/perl scripts to manage systems, parse logs, collect or push events, format reports, and issue alarms. To be effective required tribal knowledge along with proficiency in programming, network plumbing, and systems management, so changes to the environment or workforce made it extremely difficult(expensive) to deliver continuous monitoring capabilities to operators at any scale.

We are in a similar state today with network design and enterprise planning. Infrastructure-as-Code, SDN, virtualization and containerization are all critical components in modern deployments, but the glue that ties them together is largely ad-hoc, and risk evaluation is still a manual task. In order to understand the security posture of a system even before it is rolled out and SIEMs are in place, we have created a tool to facilitate the automated analysis, collection, correlation, and dissemination of the security metrics mentioned above. The necessity of such a tool is critical to evaluating the efficacy of the metrics reviewed above, and provides the foundation for ongoing research in machine learning models for secure systems planning, design, and evolution as demonstrated in Chapter 5.

3.1. Methodology

If we measure an aspect of security before and after a change takes place, then we can quantify the impact that change had on security. If we test an aspect of cyber security at regular intervals, then we can determine the rate of change for that property over time. In order to sample security measurements at regular (approaching continuous) intervals, we assert that the test apparatus must be fully automated. The necessity of such automation is critical to evaluating security metrics in a repeatable and consistent way.

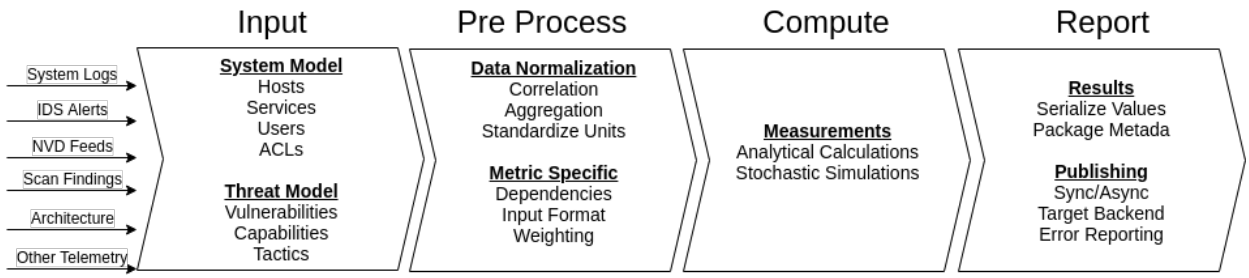


Figure 3.1: Generalized Metric Evaluation Pipeline

In Figure 3.1 we present a general four-stage pipeline for security metric processing based on our observations implementing a number of metrics from the literature. This abstraction encourages us to:

- Decouple the source of system information from the representation of that information.
- Decouple the actual calculation of the metric from the input model representation it is typically paired with in its publication.
- Decouple the calculation logic and supporting metadata from any assumptions about how that measured value will be used in the future.

In doing so, it becomes possible to identify shared dependencies among metrics, enables a systematic examination of the characteristics and behaviours of each metric across a range of inputs, and supports more reusable and composeable components for a greater variety of deployment scenarios. The remainder of this section provides the considerations and details of each stage.

3.1.1. Input Modeling

In the *Input* stage of Figure 3.1, system details depicted above the inbound arrows on the left are parsed into a model which describes the current environment or environment under test. Model parameters can be populated synthetically or from a live system. Rules comprising the threat model which describes how these components are allowed to interact with each other and with external stimulus can be added here for metrics that require it. The raw inputs to the processing pipeline can vary widely depending on which security metrics are being considered. At a high level, we treat the input stage as a black box

for handling information requests from subsequent stages. This affords us the freedom to connect static data for testing and experimentation, and live data for production deployments without altering the contract or interface. In practice input targets can be existing APIs provided by SIEMs, query interfaces to a configuration database, source code repositories, vulnerability information feeds, generated network topologies, etc. At this stage we only assume appropriate adapters exist to make this data available for the subsequent stages.

3.1.2. PTaH: Preprocessing and Transformation Handling

The *Pre Process* stage transforms the current knowledge base into a format suitable for the desired metric calculation.

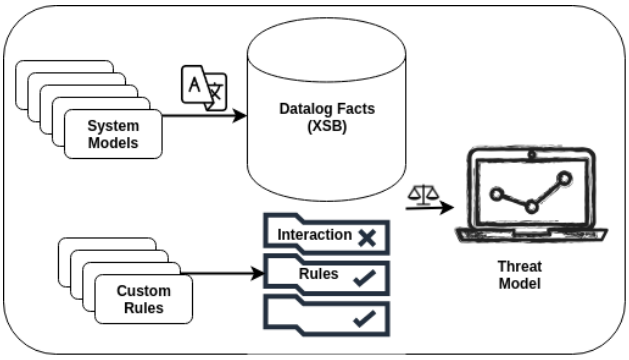


Figure 3.2: Preprocessing and Transformation Handlers (**PTaH**)

In metrics that compute aggregates, ratios, or simple statistics from findings and system facts, preprocessing steps may be minimal or bypassed entirely. In more complex metrics, such as those which consider the relationships between components or vulnerabilities, it may be necessary to craft the inputs expected from some composition of system facts along with some composition or chain of metric dependencies. In particular, metrics based on attack graphs and attack nets tend to make assumptions about the input structure which can be managed in this layer. We create these structures, score transitions, apply weights and mappings, and perform any other manipulations of our knowledge base in this layer to adhere to the input assumptions of particular metrics in this layer.

3.1.3. SecMet: A Library of Security Metrics

The *Compute* stage implements the calculation of the security metric and takes the measurement of the current state.

The surveys we covered in Section 2.2.1 describe properties common to all the security metrics they consider, which become evaluation criteria for their review. All security metrics inherit from a parent metric class that defines these common properties and the current system model, along with housekeeping functions and metadata like citations and usage.The security metric does not contain logic to create the inputs it operates on, so we can stack metrics to run in parallel against a single source of facts, or chain them in a processing pipeline to compose more complex analytics.

Our architecture for implementing security metrics is straight forward. We declare a *base security metric* type from which all metrics inherit three methods:

- Check Prerequisites: is invoked either directly by the caller or in the calculate method to ensure all items necessary for the calculation are present.
- Calculate: returns the resulting measurement
- Get Metadata: returns the environment and ancillary data used during the calculation.

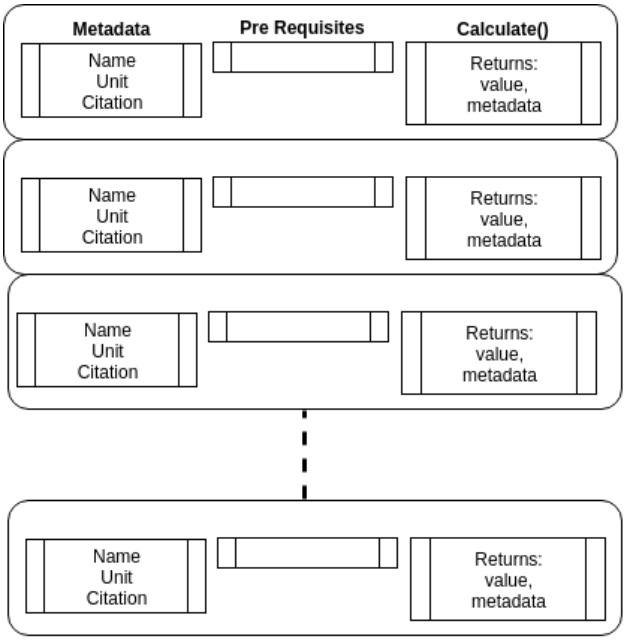


Figure 3.3: Security Metric Catalog (**SecMet**)

As shown in Figure 3.3, this design allows us to implement a library of security metrics with a standard, stateless interface.

The attack graphs described in the literature vary somewhat in structure among implementations. For example, the AGs presented in [299] include non-exploits along with exploits as nodes with edges representing lateral movements. In [292] the non-exploit nodes don't appear to be present in the publication, although the TVA tool isn't publicly available to test this. In [108] and [297] nodes represent system privileges and edges contain exploits that grant an attacker additional privileges on a set of systems, while in [315] edges carry probabilities of exploitation and the nodes represent actual hosts. While the differences are subtle, they are enough to necessitate a general form of attack graph which we present in Figure 3.6. Our representation of an attack graph is a multi-edged directed acyclic graph.

As shown in Figure 3.7, our implementation allows us to load various AG formats from graph description language (.dot) files, from adjacency lists as shown in Tables 2.2a and 2.2b, or other formats as specified. Once loaded, we provide programmatic access to manipulate scoring and weighting functions, exploits definitions, and vulnerability scores. This allows for a simple means to test the range of values a metric will generate, the sensitivity of a metric to fluctuations in parameters, and how well a metric performs on different models. It also gives us some insights into how well each AG type actually models threat and defense attributes.

3.1.4. Measurement Reporting

Finally, the *Reporting* stage handles the response logic needed to return the measured value and associated metadata appropriately. In our experience, most security metrics return a single value, although heuristics are also supported as bucket sizes and value count arrays returned within the accompanying metadata. In these cases we return a value of -1 and a unit of the type to expect in the metadata, eg array or matrix. The metadata['value'] key then points to the resulting complex structure.

3.1.5. Security Metric Validation

In [415] Verendel finds 4 distinct validation methods used across the 90 security metrics papers surveyed: hypothetical, empirical, simulation, and theoretical. The author adds the caveat that no attempt was made to verify the quality of results, only to describe the methods used in each paper to substantiate the findings. In this section we propose a method for validating security metrics through empirical means.

In general, a metric should be both reliable and valid, where reliability refers to the consistency of values across repeated measurements, and validity concerns the accuracy of those values. There is currently no *unit* reference for a security property which we can use for establishing the accuracy of our security metrics. We can, however, examine the behaviour of a metric relative to a given system by manipulating relevant aspects of that system while holding the other properties constant. For example, we can assign vulnerabilities to the small enterprise network shown in Figure 3.4a. For metrics that are influenced by the vulnerability score (such as CVSS based metrics), we can fix the weights of these vulnerabilities in the range of possible scores (0.0 to 10.0 in the case of CVSS). This would produce the lower and upper bound for that metric on this specific configuration of system configuration and vulnerability assignment. Similarly, we can alter properties such as the effects of a successful exploit (remote code execution, privilege escalation, etc), the placement and quantity of vulnerabilities, the underlying platform/operating system/applications, connectivity and topology, and so on. By capturing the measured values of controlled alterations to a system, we begin to understand how that metric behaves on the given system. Observing how multiple metrics behave under the same alterations allows us to compare their relative performance without an absolute point of reference. Furthermore, we can extend this analysis from a simple enterprise network to any number of use cases by supplying input adaptors for topology generators such as BRITE[257] or simulators like SSFNet[103] and Mininet[222]. Figure 3.4 shows an example reference set of 3 network topologies of increasing size and complexity. The first represents the type of example network commonly referenced in the literature to demonstrate a newly proposed metric, and the other two were selected from the

gallery of publicly available SSFNet topologies and are described by an easy to parse domain specific language. By including systems at different scales, we are able to determine if a metric's performance depends on the size and complexity of the system under test. Intuitively, we would expect the temperature of 1 liter of boiling water to be the same as 1000 liters of boiling water, so testing against systems of varying size allows us to verify this hypothesis.

If we consider a security metric as a measurement instrument, then we can validate the metric by characterizing its behavior against a reference set. Some security metric properties we might consider for validation can be taken directly from the field of measurement theory. In addition to accuracy and precision, Morris[276] describes several performance characteristics of measurement instruments that we may adapt to our validation of security metrics:

Monotonicity: Does the measured value always increase or always decrease with respect to each improvement in security.

Linearity: For security levels s_1 and s_2 and incremental security improvement i, is the difference between (s_1, i) the same as (s_2, i)?

Range: What are the upper and lower bounds that a metric will assume for a specific model?

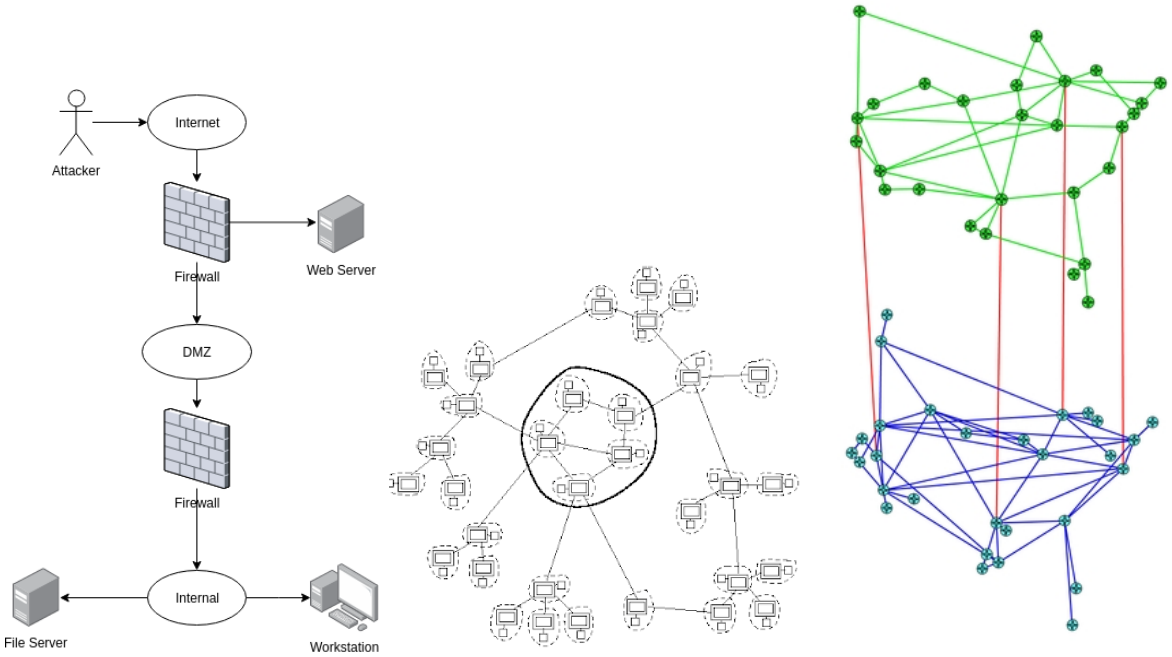
Precision: Does the metric report the same value for repeated measurements of the same quantity? This is of particular interest in probabilistic metrics.

Analogues to other standard instrument performance measures - threshold, resolution, sensitivity to disturbance, etc - can be adapted to evaluate a security metric against our reference networks as well, and it is still to be determined which characteristics of static measurement instruments are required for their security related counterparts.

3.2. Implementation

Our architecture for implementing security metrics is straight forward. We declare a *base security metric* type from which all metrics inherit three methods:

- CheckPreReqs: is invoked either directly by the caller or in the calculate method to



(a) Small-sized network[299] (b) Medium-sized network[103] (c) Large-sized network[103]

Figure 3.4: Different Sized Reference Networks

ensure all items necessary for the calculation are present.

- Calculate: returns the resulting measurement
- GetMetadata: returns the environment and ancillary data used during the calculation

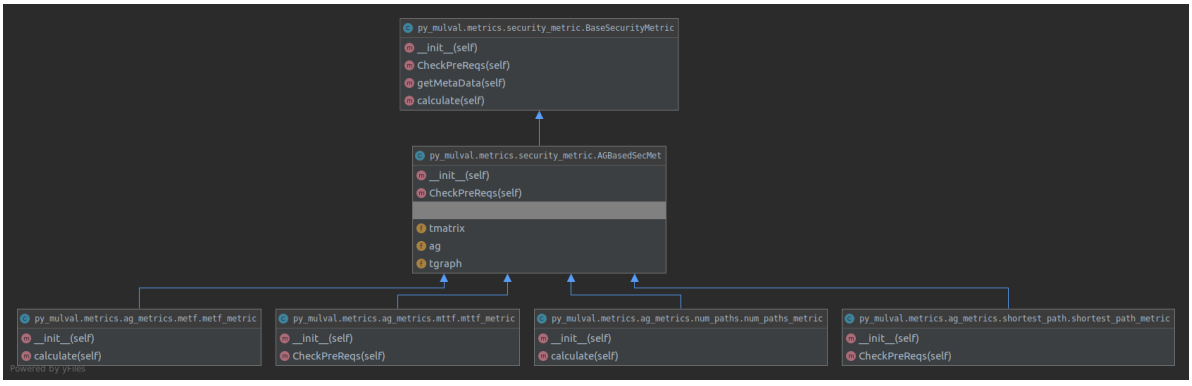


Figure 3.5: Metric Class Diagram

Figure 3.5 shows the inheritance hierarchy for attack graph based security metrics. Each AG metric implementation inherits from AGBasedSecMetric which in turn inherits from BaseSecurityMetric. AGBasedSecMetric includes a property for attack graph which is a

requirement to perform the metric calculation. Each BaseSecurityMetric also contains immutable properties for the name of the metric, the expected measurement unit for results, a citation field linking to the source paper where we found the metric, and a short summary of what the metric does. All these properties plus any additional information derived during calculation are added to the metadata dictionary of the metric and returned when the calculate method is called.

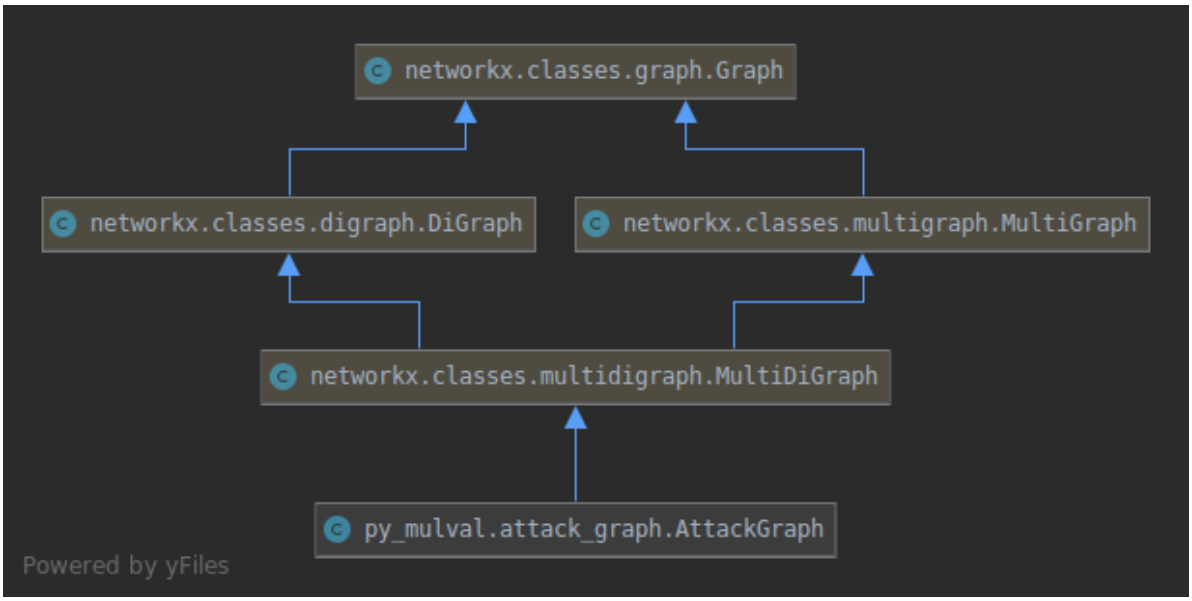


Figure 3.6: Attack Graph Class Diagram

The attack graphs described in the literature vary somewhat in structure among implementations. For example, the AGs presented in [299] include non-exploits along with exploits as nodes with edges representing lateral movements. In [292] the non-exploit nodes don't appear to be present in the publication, although the TVA tool isn't publicly available to test this. In [108] and [297] nodes represent system privileges and edges contain exploits that grant an attacker additional privileges on a set of systems, while in [315] edges carry probabilities of exploitation and the nodes represent actual hosts. While the differences are subtle, they are enough to necessitate a general form of attack graph which we present in Figure 3.6. Our representation of an attack graph is a multi-edged directed acyclic graph.

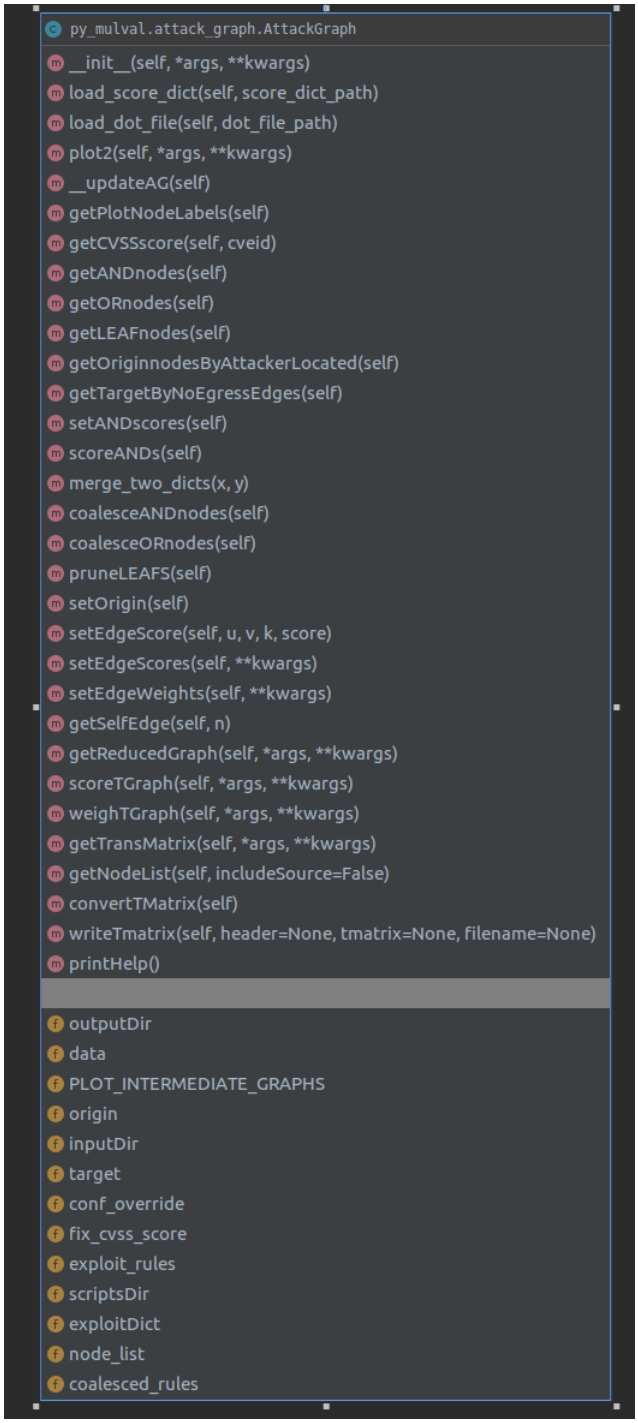


Figure 3.7: Attack Graph Methods and Properties

As shown in Figure 3.7, our implementation allows us to load various AG formats from graph description language (.dot) files, from adjacency lists as shown in Tables 2.2a and 2.2b, or other formats as specified. Once loaded, we provide programmatic access to manipulate

scoring and weighting functions, exploits definitions, and vulnerability scores. This allows for a simple means to test the range of values a metric will generate, the sensitivity of a metric to fluctuations in parameters, and how well a metric performs on different models. It also gives us some insights into how well each AG type actually models threat and defense attributes.

In the *Input* stage of Figure 3.1, system details depicted above the inbound arrows on the left are parsed into a model which describes the current environment or environment under test. Model parameters can be populated synthetically or from a live system. Rules comprising the threat model which describes how these components are allowed to interact with each other and with external stimulus can be added here for metrics that require it. The raw inputs to the processing pipeline can vary widely depending on which security metrics are being considered. At a high level, we treat the input stage as a black box for handling information requests from subsequent stages. This affords us the freedom to connect static data for testing and experimentation, and live data for production deployments without altering the contract or interface. In practice input targets can be existing APIs provided by SIEMs, query interfaces to a configuration database, source code repositories, vulnerability information feeds, generated network topologies, etc. At this stage we only assume appropriate adapters exist to make this data available for the subsequent stages.

The *Pre Process* stage transforms the current knowledge base into a format suitable for the desired metric calculation.

In order to experiment with different network architectures and exploit effects, it was necessary to reproduce each step in the process: Install AG suite (MulVal, XSB) → Define AG input model (Datalog/Prolog) → Generate AG (Java, C++, ANTLR4, sed) → Import NVD (JSON, XML → SQL) → Implement custom adaptor for Stochastic Model expected input (Python, inference) → Insert transition matrix into provided R script (ctrl+c, ctrl+v) After following this process precisely 2 times we understood why the AG based analytics community isn't larger. Our immediate solution was to create a set of ansible roles and plays to automate the environment setup, test execution, and results collection entirely

with a single command. This in itself lowers the barrier to entry for anyone interested in experimenting with attack graphs or looking to (quickly) reproduce our results.

To handle the scale and volume of requests needed to support the advanced use-cases listed below, we are currently implementing and evaluating the following features in the SMaaS architecture: Metric Isolation: Each metric should be independently deployable to allow scaling up and down as request volume dictates. Currently metrics are bundled in Python and R modules with logical separation at the function level.

The *Compute* stage implements the calculation of the security metric and takes the measurement of the current state.

The surveys covered in Section 2.2 describe properties common to all the security metrics they consider, which become evaluation criteria for their review. All security metrics inherit from a parent metric class that defines these common properties and the current system model, along with housekeeping functions and metadata like citations and usage.The security metric does not contain logic to create the inputs it operates on, so we can stack metrics to run in parallel against a single source of facts, or chain them in a processing pipeline to compose more complex analytics.

Finally, the *Reporting* stage handles the response logic needed to return the measured value and associated metadata appropriately. In our experience, most security metrics return a single value, although heuristics are also supported as bucket sizes and value count arrays returned within the accompanying metadata.

3.2.1. Interaction Rules for Infrastructure

The system model used by MulVal abstracts network infrastructure into a set of access control rules of the form `hacl(_src, _dst, _prot, _port)`. This `hacl/4` is a MulVal atomic *primitive*, which to us means that it should be provided as input to the run. Horn clauses[86] of the form $A \leftarrow B_1, B_2, \ldots, B_n$ are the foundation of MulVal's reasoning system. *Facts* (primitive or derived) on the right-hand-side specify a relationship described by the term on the left-hand-side. *Interaction Rules* in the framework wrap the Horn clause in

a tuple with description and labeled weight metadata needed for attack graph processing and rendering. We use the terms interchangeably within this paper unless otherwise noted, and present the unwrapped form when discussing interaction rules for clarity. The following Horn clause derives facts about network accessibility:

```
netAccess(P, H1, H2, Protocol, Port) :- execCode(P, H1 _User),
                                        hacl(H1, H2, Protocol, Port).
```

This statement can be read: *Principal **P** has network access from host **H1** to host **H2** on **Port** using **Protocol** if **P** can execute code on **H1** as **_User** and **H1** has access to **H2** over that **Port** and **Protocol**.* An interaction rule provides results for a single query on a relationship. A logic program is a collection of Horn clauses sufficient to cover a desired query space. In MulVal, `meta(attackGoal(_))` is defined up front as the program objective, with the query `findall(Goal, attack_simulation(Goal), L)` responsible for collecting attack traces.

```
1  primitive(inCompetent(_principal)).
2  primitive(competent(_principal)).
3  primitive(clientProgram(_host, _programname)).
4  primitive(vulExists(_host, _vulID, _program)).
5  primitive(vulProperty(_vulID, _range, _consequence)).
6  primitive(hacl(_src, _dst, _prot, _port)).
7  primitive(attackerLocated(_host)).
8  primitive(hasAccount(_principal, _host, _account)).
9  primitive(networkServiceInfo(_host, _program, _protocol, _port, _user)).
10 primitive(setuidProgramInfo(_host, _program, _owner)).
11 primitive(nfsExportInfo(_server, _path, _access, _client)).
12 primitive(nfsMounted(_client, _clientpath, _server, _serverpath, _access)).
13 primitive(localFileProtection(_host, _user, _access, _path)).
14 primitive(dependsOn(_h, _program, _library)).
15 primitive(installed(_h, _program)).
16 primitive(bugHyp(_,_,_,_)).
```

```
17 primitive(vulExists(_machine,_vulID,_program,_range,_consequence)).
18 primitive(canAccessFile(_host, _user, _access, _path)).
19 primitive(isWebServer(_host)).
20
21 derived(execCode(_host, _user)).
22 derived(netAccess(_machine,_protocol,_port)).
23 derived(canAccessHost(_host)).
24 derived(accessFile(_machine,_access,_filepath)).
25 derived(accessMaliciousInput(_host, _principal, _program)).
26 derived(principalCompromised(_victim)).
27 derived(dos(_host)).
28 derived(logInService(_host, _protocol, _port)).
```

Listing 3.1: Mulval Primitive and Derived Facts

There is no explicit definition for network infrastructure in MulVal, but there is also no syntax requirement to adhere to the variable naming semantics. Remember that Prolog variables begin with either an upper case letter, or an underscore ('_'), and that constants begin with lower case letters. Thus we are free to overload the existing `hacl/4`, and `netAccess/4` predicates to define the Layer 2 and 3 network elements for our desired system model.

```
hacl(sw1_xe_1_0_2, sw2_xe_1_1_1, vlan, ac_10_20).
hacl(sw1_et_1_0_1, sw_et_1_0_2, vlan, tr_30).
hacl(st2_ge_1_0_2, sw1_ge_1_1_1, vlan, mgmt_1).
hacl(rt2_ge_1_1_2, _, bgp, 179).
hacl(rt2_ge_1_1_2, ospf_grp_1, ospf, 179).
hacl(rt3_ge_1_1_2, ospf_grp_1, ospf, 179).
```

We are aware of efforts[40, 168] to define these network elements as extensions to MulVal; however, we find the existing Datalog structures sufficient for our analysis without modifying the underlying model. We briefly discuss our reasoning and refer to the core and edge network designs found in [125] for reference.

MulVal is primarily concerned with application layer attacks, with the primary effects of a successful attack being remote code execution and privilege escalation. In core and transit networks, the target is typically DoS, hijacking, or eavesdropping.The types of attacks we intend to model influence our syntax decisions.

For example, we can describe switches and routers along with their associated protocols without modification of the existing MulVal rules. This makes intuitive sense since the distinction between these devices becomes less clear when their functionality is virtualized. Having a VM running OpenVSwitch talking to a Layer 3 physical switch over Spanning Tree Protocol shouldn't require separate definitions from Layer 2 switches participating in the same protocol. Note this doesn't imply the attack surfaces are the same, and in Secion 2 we make this distinction explicitly. In the examples above we show how existing `hacl/4` primitives can be used to specify VLAN tagged switch ports and OSPF areas for routers ports. The general naming convention to describe elements is underscore delimited, with the first entry describing the element (rt: router, sw: switch, rr: route reflector, ce: customer edge, pe: provider edge, p: provider core). Remaining fields identify the line card, interface, and port uniquely in a device and can be inherited from the organizations standard nomenclature.

MulVal assumes attacker privilege is monotonically increasing, meaning that through the course of an attack an intruder can not decrease system access through vulnerability exploitation. This is a known[299, Ch 2.6] effect inherited from Datalog and largely precludes the study of denial-of-service type attacks using the CSAF. That said, the types of attacks we can model using existing constructs still cover a broad range of Layer 2 and Layer 3 vulnerability classes which we define specifically in Section 2.

Defining the router interfaces as unique `host` entries with distinct `hacl/4` parameters permits us to treat exploit consequences like we would any other system. As an example, consider a provider edge router with an improperly configured access control list. If an attacker located within the attached customer edge network can discover this configuration error, the model reflects the new `netAccess/5` state. In some tests we declare control, data,

and management plane interface groups for each device to represent interface groups for simplicity. This naively represents the functionality an attacker would potentially target. By modeling the infrastructure devices this way we can define services on any plane as `networkService/5` and express a compromise leading to unauthorized access of a different plane clearly for use in our trace analysis.

3.2.2. Development & Testing Environment Setup Automation

One of the difficulties we encountered when trying to reproduce results from other papers was setting up the environment and running a test quickly. We created a set of ansible[160] roles and playbooks to provision the environment and automate executing experiment pipelines for multiple input models. The roles we created set up a clean environment with the attack graph engine and dependencies installed. In some cases the components are no longer actively maintained, so automating the deployment with ansible provides a reproducible environment that allows us to fix dependencies at the required version, track patches necessary to accommodate ongoing updates to linked projects, and maintain and deploy custom rules all under version control. The playbooks we developed allow us to customize the analysis pipeline by simply including modules for each step in the process. As shown in Listing 3.2, task lists representing modules can be as granular as necessary. Execution dependencies are defined by tags, allowing us to specify run targets and requirements selectively. For example, we can avoid duplicating time consuming tasks like downloading and preparing the CVSS DB, set the environment to a known state for development and debugging, or prepare and deploy results for consumption by other tools such as SIEM systems or monitoring dashboards.

```
1   hosts: all
2 gather_facts: True
3 vars_files:
4   vars/main.yml
5 roles:
6   role: ansible role mysql # NVD CVSS DB
7   role: ansible role mulval # setup MulVal, XSB, etc
8 tasks:
9   include_tasks: setup.yml # deploy model,update NVD
10  tags: [models, run, db]
11  include_tasks: run.yml # generate attack graphs
```

```yaml
---
- hosts: all
  become: no
  #strategy: debug
  gather_facts: True

  vars_files:
    - vars/main.yml

  roles:
    - role: ansible-role-mysql
      become: True

    - role: ansible_role_mulval
      become: False

  tasks:
    # - deploy Mulval models to inventory hosts
    # - update NVD CVSS db
    - include_tasks: setup.yml
      tags: [models,run,db]
    # - generate attack graphs for deployed models
    - include_tasks: run_mulval.yml
      tags: run
    # - create transition matrices for attack graphs
    - include_tasks: gen_trans.yml
      tags: run
    # - setup analysis environment (R, py, Octave,...)
    - include_tasks: setup_analysis.yml
      tags: run
    # - run analytics
    - include_tasks: run_analysis.yml
      tags: run
    # - format results, collect graphs, ...
    - include_tasks: finalize.yml
      tags: run
```

Figure 3.8: CSAF playbook

```
12    tags: [run, cleanup]
```

Listing 3.2: Ansible play

When preparing the target system, a directory containing the MulVal models to analyze is expected. These models are copied to the target system in individual directories since we can't control the output file names chosen by MulVal without modifying the distribution.

When preparing the target system, a directory containing the MulVal models to analyze is expected. These models are copied to the target system in individual directories as shown in Listing 3.3 since we can't control the output file names chosen by MulVal without modifying the distribution.

```
13    name: copy models to remote in individual directories (mulval output is noisy)
14  copy:
15  src:  "{{item.path | relpath('{{mulval_models_src}}')}}"
16  dest:  "{{mulval_models_dir}}/{{(item.path|basename|splitext)[0]}}"
17  with_items: "{{find_results.files}}"
18  tags: [models, run]
```

Listing 3.3: MulVal distinct run dirs

MulVal supports a single custom interaction rules file being passed as a runtime argument. To allow for a more customizable custom rules set, we allow for the rules defined per model or globally as shown in Listing 3.4. Assuming the custom rules are defined in separate files in the *mulval_custom_rules_dir* directory, we can provide a list of rules for *each* model we are testing, concatenate *all* rules in the directory for every test, or omit custom rules for the test altogether.

```
19   name: run mulval (all rules)
20 shell: bash lc "{{mulval_home}}/utils/graph_gen.sh {{ item.path|basename }} -p -v -a {{
       custom_rules_dir}}/custom_rules.P"
21 args:
22 chdir: 'oo"{{(item.path|dirname)}}"
23 with_items: " {{ find_results.files }} "
24 tags: run
25 when: custom_rules_strategy == 'all'
26   name: run mulval (each rule)
27 shell: bash lc "{{mulval_home}}/utils/graph_gen.sh {{ item.path|basename }} -p -v -a {{
       custom_rules_dir}}/{{ item.path|basename }}.rules "
28 args:
29 chdir: "{{(item.path|dirname)}}"
30 with_items: " {{ find_results.files }} "
31 tags: run
32 when: custom_rules_strategy == 'each'
33   name: run mulval (no rules)
34 shell: bash lc "{{mulval_home}}/utils/graph_gen.sh {{ item.path|basename }} -p -v"
35 args:
36 chdir: "{{(item.path|dirname)}}"
37 with_items: " {{ find_results.files }} "
38 tags: run
39 when: custom_rules_strategy == 'none'
```

Listing 3.4: Custom rule strategies

Once processing has completed, results are collected by simply transferring them to the desired location. Adding enhanced reporting and alerting capabilities as new results arrive can be achieved using a variety of monitoring tools (eg., inotify, db triggers, fluentd) depending on the destination. As shown in Listing 3.5. Adding enhanced reporting and alerting capabilities as new results arrive can be achieved using a variety of monitoring tools (eg., inotify, db triggers, fluentd) depending on the destination.

```
40    name: fetch results (get them all )
41  synchronize:
42  mode: pull
43  src: "{{mulval_results_dir}}"
44  dest: "{{mulval_output_dir}}"
45  #with_items: "{{ find_results.files}}"
46  tags: [ run, collect ]
```

Listing 3.5: Collect Results

3.2.3. Graph Reduction

MulVal produces multiple output files when it generates an attack graph. The following section documents the process used to generate the transition matrix required for input to the CSAF using the output from MulVal. Running MulVal with the `-l` or `-v` option will create the necessary output files.

Require: ARCS.CSV and VERTICES.CSV
  for vertex in VERTICES.CSV **do**
    parseVertex() {store NodeID, NodeType, and NodeText}
  end for
  for arc in ARCS.CSV **do**
    parseArc() { store immediate predecessors and successors}
  end for
  n = count(orNodes) {*only include exploitable OR nodes}
  tmatrix = zeroes(n,n) {initialize nxn matrix with 0's}
  for each OR node **do**
    for each parent OR node **do**
      tmatrix[parentOR][currentOR]= exploit success probability
    end for
    tmatrix[currentOR][currentOR]= exploit failure probability
  end for
  return TransitionMatrix.csv

Algorithm 1: Calculate Transition Matrix

To demonstrate, we calculate the transition matrix for the simple attack graph example included with the MulVal package and documented in Section 2.1.3. We adopt the loop removal approaches put forth in [300] to discard cycles in the attack graph before translating to the transition matrix. Logic reduction techniques have been identified in [175] to reduce the state space of the attack graph. Our implementation follows the methodology implied in [9] by coalescing nodes that are reachable post-exploit without requiring further compromise

(so called 'multihop access' nodes) to further reduce state space. Also, while the effects of various CVSS based weighting schemes are explored in [362], our design uses a simple normalized average when assigning probabilities among multiple outbound vulnerabilities.

Tables 2.2a and 2.2b give the MulVal output generated for the attack graph nodes and edges shown in Figure 3.9. The resulting unweighted transition matrix is shown in Table 3.1a. This is a square nxn matrix which contains only OR nodes that can be reached through successful exploitation. Referring again to Figure 3.9a:

1. Node 0 is the attacker's origin
2. From node 0, only node 13 is reachable
3. From node 13, either node 10 or node 5 can be reached
    - Node 10 is immediately available given the attacker's current privilege level, so no exploit is necessary. We coalesce this node in the path between node 13 and node 8 in the transition matrix.
    - Node 5 is directly reachable from node 13 and node 8
4. Node 3 is only reachable from node 5
5. Node 1 (target) is only reachable from node 3

Table 3.1: Transition Matrix

NodeID	0	13	8	5	3	1
0	1	1	0	0	0	0
13	0	1	1	1	0	0
8	0	0	1	1	0	0
5	0	0	0	1	1	0
3	0	0	0	0	1	1
1	0	0	0	0	0	1

(a) unweighted transition matrix

0	13	8	5	3	1
0	1	0.0	0.0	0.0	0.0
0.0	0.24	0.22	0.55	0.0	0.0
0.0	0.0	0.29	0.71	0.0	0.0
0.0	0.0	0.0	0.62	0.38	0.0
0.0	0.0	0.0	0.0	0.5	0.5
0.0	0.0	0.0	0.0	0.0	1.0

(b) weighted transition matrix P

We see the unweighted transition matrix in Table 3.1a is a direct transcription of the vulnerable nodes in the attack graph and their reachability. In the next step we assign probabilities of successful exploitation for use in simulations, so the diagonal is populated

with 1's here to hold the probability an attacker fails to compromise the subsequent step and remains at the current state.

Following the simple weighting strategy identified in [9] we let the probabilities of the transition matrix P be determined by $p(i,j) = \frac{e_j}{\sum_{k=1}^{n} e_k}$ where n is the number of outgoing edges from a given state i, and j is the exploitability score for a vulnerability in state j determined by its CVSS value.

In the event a CVSS score is unavailable in the NVD, we provide several mechanisms to control the weight assigned to transitions by taking advantage of the 'hypothetical vulnerability' capabilities built into MulVal. For example, 'CAN-2002-0392' uses the (now deprecated) CANdidate prefix for the 'CVE-2002-0392' vulnerability and is not found in the current NVD database sync. Often products with a large user base will maintain their own list of public vulnerabilities that don't get assigned a CVE identifier. So, to specify individual vulnerability identifiers and scores (or to override existing ones) we can load a dictionary of vulnerabilities from configuration that will be checked before looking up values in the local NVD instance. We also allow default scores for individual vulnerability classes (eg 'Trojan horse installation') and the ability to define a catchall score or 'alert and exit' mechanism if no matching vulnerability is found. For the following manually entered exploitability scores we obtain the weighted transition matrix found in Table 3.1b:

- 'CAN-2002-0392' = 7.5
- 'direct network access' = 10
- 'NFS shell' = 9.5
- 'execCode implies file access' = 7.8
- 'NFS semantics' = 9.6
- 'Trojan horse installation' = 5
- 'vulID' = 1 # fallback value for undefined vulnerabilities

Table 3.2: Weighted transition matrix P

15	13	8	5	3	1
0.571	0.428	0.0	0.0	0.0	0.0
0.0	0.416	0.055	0.527	0.0	0.0
0.0	0.0	0.113	0.886	0.0	0.0
0.0	0.0	0.0	0.473	0.526	0.0
0.0	0.0	0.0	0.0	0.657	0.342
0.0	0.0	0.0	0.0	0.0	1.0

Starting at the top row, from node 15 (direct internet access) an attacker can attempt to exploit node 13 (remote code execution). The probability the attack is successful is $\frac{7.5}{7.5+10} = 0.42857\ldots$ and the probability the attack fails is $1 - \frac{7.5}{7.5+10} = 0.5714\ldots$ Obviously care should be taken when assigning exploitability scores to hypothetical vulnerabilities to yield a sufficiently realistic view of the network security posture. We have designed the framework to be configurable both in the weights assigned to vulnerability classes and to the weighting strategy in general as described in Section 3. The portions of this paper implementing CSAF directly use the weighting strategy described in here, although the benchmarking results presented in Section 4.5.1 find surface some issues with this strategy.

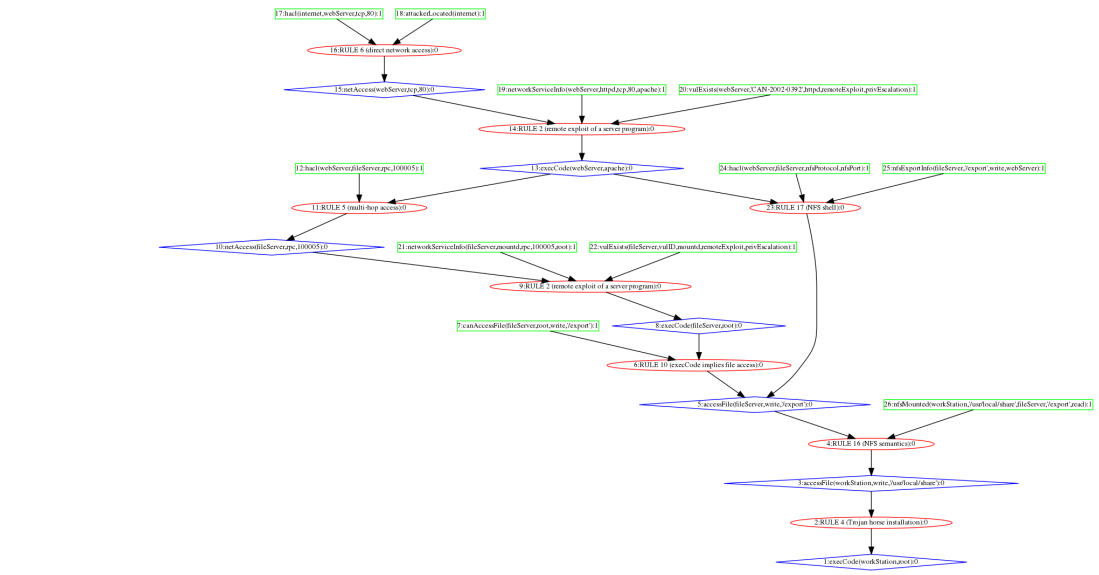
3.3. Results

In this section we describe our proof of concept implementation of the pipeline shown in Figure 3.1 and demonstrate how it can be used to address some current issues in security measurement research. We provide an example environment for metric development and evaluation. We create a library of select security metrics from the literature and build up reusable processing components for end-to-end automation of the pipeline. We then demonstrate the utility of the framework with two motivating examples. In the first case we develop a simple validation methodology for security metrics. Then we describe a distributed stream processing architecture for deploying security metrics in production, and discuss practical considerations for scaling, securing, and managing dependencies in the metric pipeline.. The

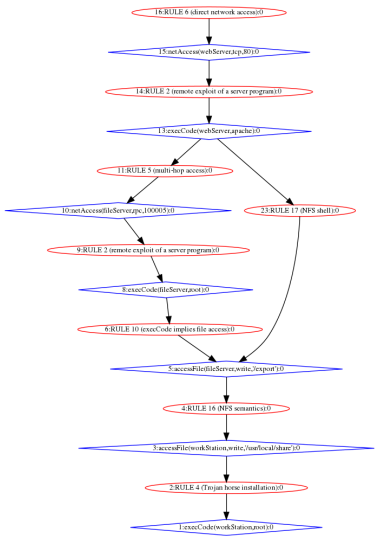
entire workflow can be thought of in terms of the well-known *extract, transform, load* (ETL) process, which opens up a variety of design, implementation, and deployment options. Our goal in this section therefore is to describe contributions specific to security metrics analysis, where adherence to the literature is prioritized above computational optimization and efficiency. In Section 3.2.2 we described how to use ansible as a driver to control process flows which easily creates a clean environment suitable for development and testing of pipeline components like interaction rules, transform logic, metrics, and reporting formats. In Section 3.3.1 we discuss how implementing this pipeline as a (Python) library enables us to batch experiments for controlled validation and analysis. Finally, we review the details of how we have implemented the metrics catalog and some findings we have come across regarding their presentation in the literature.

3.3.1. Processing Pipeline Instrumentation

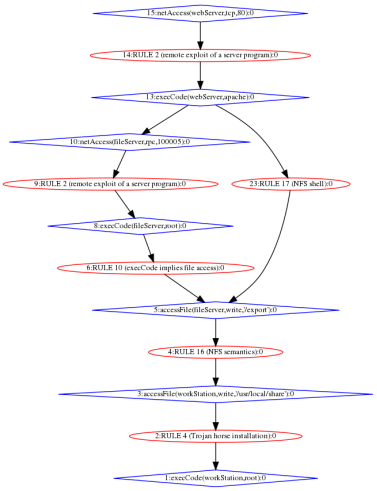
Originally our focus was on reproducing graph based security metrics. While we have since expanded the scope to include the range of metrics that can be applied to any area of cyber security, the metrics which exploit the relationships between components in the system typically require the most preprocessing to shape the inputs correctly. For instance, a count of the number of critical vulnerabilities found from a scan is a straight forward metric calculation. Determining which vulnerabilities are reachable in a given network topology requires more effort, and ensuring that reachability graph conforms to the assumptions of the target metric (non-invertable weighted transition matrix or some other specific representation) requires still further computation. Thus, PTaH exposes some general statistical methods for the simpler metrics as well as some more fine grained helpers that apply to subsets or individual metric implementations. In this way we can maintain the simple abstractions of both the raw input source and the actual metric calculation, while also supporting the preprocessing requirements for new metrics. As mentioned above, there are many ETL pipeline implementations available which would be suitable for this implementation, so in this section we demonstrate examples of metric specific data transformations rather than the underlying



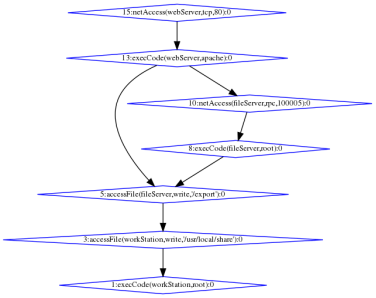
(a) Attack graph produced by MulVal



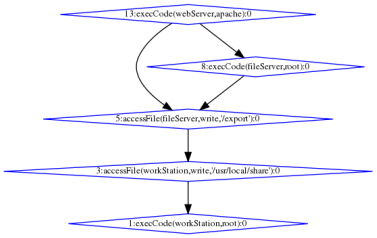
(b) LEAFs scored & pruned



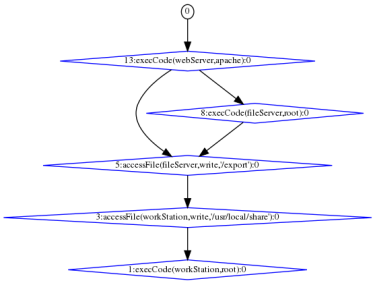
(c) Merge non-exploit ANDs



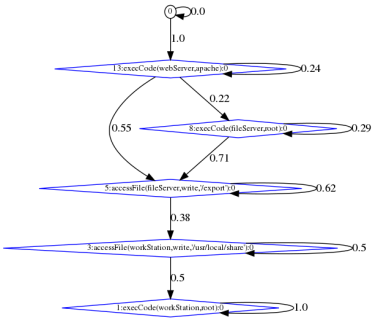
(d) Merge remaining ANDs



(e) Merge remaining ORs



(f) Add pseudo-root



(g) Calculate advance probs

Figure 3.9: Attack Graph Reduction

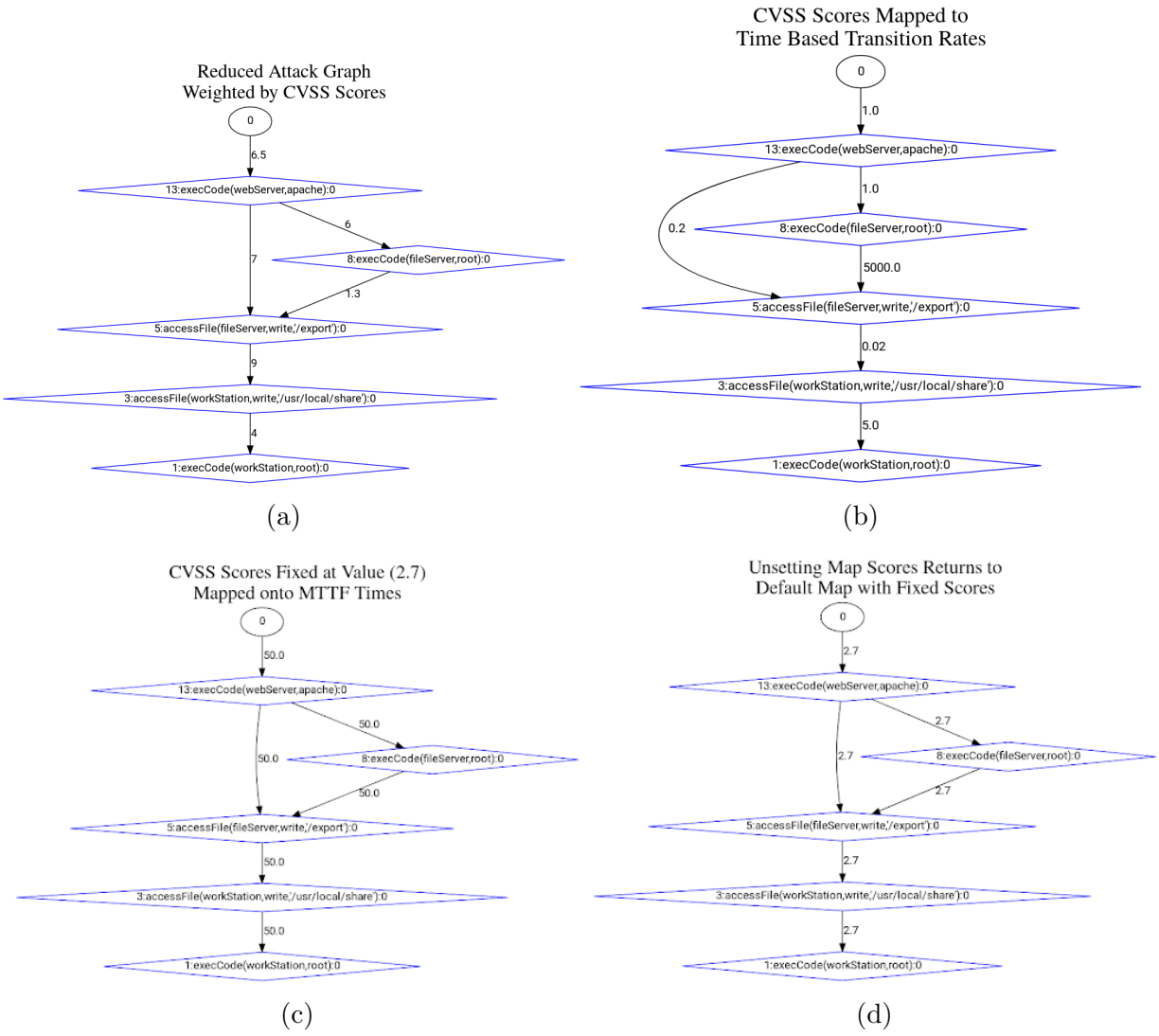


Figure 3.10: Graph Weight Manipulation: These pairs should have equivalent security for the respective target metrics.

technology used to execute them.

The attack graph models described in the literature vary somewhat in structure among implementations. For example, the AGs presented in [299] include non-exploits along with exploits as nodes with edges representing lateral movements. In [292] the non-exploit nodes don't appear to be present in the publication, although the TVA tool isn't publicly available to test this. In [108] and [297] nodes represent system privileges and edges contain exploits that grant an attacker additional privileges on a set of systems, while in [315] edges carry probabilities of exploitation and the nodes represent actual hosts. While the differences are

subtle, they do require an understanding of the target input expected by each metric. In these cases, our solution is to provide helper methods that transform the provided knowledge base into any of these expected input formats. If an attack graph is needed for processing, relevant system information is translated into datalog facts for use in MulVal[299]. MulVal consists of 2 primary components, an XSB Datalog program that computes attack traces from a given set of facts and interaction rules, and a C++ program that processes the XSB output trace and produces an annotated list of vertices and edges corresponding to possible attack paths. To allow more control over MulVal processing we replaced the driver shell script with a python interface, allowing programmatic access to the facts and interaction rules that are used to derive attack graphs directly over a Python-XSB bridge. This allows us to manipulate structural properties of the input system and delegate vulnerability placement strategically while avoiding many of the high cost disk operations that the original controller required.

While the differences are subtle, they are enough to necessitate a general form of attack graph which we present in Figure 3.6. Our representation of an attack graph is a multi-edged directed acyclic graph.

In Figure 3.9, we show the process of ingesting an attack graph produced by MulVal[299] and transforming it into a normalized weighted graph for export to a dense transition matrix used in several probabilistic security metrics. A MulVal attack graph contains three types of nodes. Referring to Figure 3.9a, *LEAF* nodes (green rectangles) describe known facts like configuration information and attacker privilege that are given as inputs to the system model. Internal nodes generally represent potential privileges to be gained by an attacker. *AND* nodes (red ovals) contain interaction rules that dictate which facts and conditions are necessary to derive new knowledge. *OR* nodes (blue diamonds) represent derived facts such as transition states possible given all incoming conditions are satisfied. Each step in the reduction process can be called explicitly via API to produce the desired result, or the composition of a series of individual transforms can be exposed as a single call.

Once loaded, we provide programmatic access to manipulate scoring and weighting func-

tions, exploit definitions, and vulnerability assignments. This allows for a simple means to test the range of values a metric will generate, the sensitivity of a metric to fluctuations in parameters, and how well a metric performs on different models. It also gives us some insights into how well each AG type actually models threat and defense attributes. While we focus on attack graphs in this paper, the process for manipulating other threat model structures (such as attack trees or nets) is similar, where the goal is to introduce controlled perturbations in the representation without altering its fundamental structure to assess the behaviour of the target metric.

```
47 coalesce_rules:  # list of AND rules to coalesce
48   - 'multi-hop access by gateway'
49   - 'multi-hop access'
50   - 'direct on-host access'
51   - 'direct network access'
52   - 'log in for ftpd'
53   - 'Access a host through a log-in service'
```

Listing 3.6: Node Labels to Coalesce

The dictionaries shown in Listings 3.6, 3.7, and 3.8 provide granular control over the transformation process and can be defined statically at compile time via yaml syntax (shown), or generated dynamically at runtime and set as properties of the associated object being transformed.

The coalesce rules shown in Listing 3.6 define which types of transitions should be ignored in the final result. If we consider an adjacency matrix M where the connections between two vertices represent the likelihood of successful exploit, then in cases where no exploit was used, how is the transition represented? Leaving the value at zero may result in a non-invertable or improperly weighted result, so we simply merge any transitional edges that occur as a result of one of the rules in this dictionary.

```
54 exploit_rules:  # dict of AND rules to add to tmatrix
55   'remote exploit of a server program': 3
56   'Trojan horse installation': 4
57   'local exploit': 5
58   'NFS shell': 7
59   'execCode implies file access': 8
60   'NFS semantics': 9
61   'any machine he has an account on will also be compromised': 9.2
62   'through a log-in service': 9.3
63   'password sniffing through spoof': 4
64   'password sniffing through route hijack': 5.6
```

Listing 3.7: Fixed Scores for Exploit Classes

Similar in behavior to the coalescing rules above, we can also fix transition scores of classes of transitions based on the interaction rule they fire. This allows us to test the effects of changing the exploitability or impact scores on classes of vulnerabilities based on the outcome of that successful exploit without needing to identify every possible vulnerability that produces that outcome.

```
65 exploitDict: # dict of LEAF CVSS overrides
66   'CVE-2014-9796': 6.1
67   'CVE-2009-2048': 6.2
68   'CVE-2015-7501': 6.3
69   'CVE-2016-xxxx': 1.4
70   'CAN-2002-0392': 6.5
71   'CVE-2010-2784': 6.6  # no sql cnxn
72   'CVE-2014-6271': 6.7
73   'arpSpoofVuln': 6.8
74   'vulID': 6
```

Listing 3.8: Override Specific Vulnerability Scores

Listing 3.8 allows us to fix any specific vulnerability score individually. The override precedence favours specific over general, so individual scores set here will be applied after scoring based on exploit rules and globally fixing scores. Finally, a default scoring strategy which supports dataset queries such as NVD, or returning a fixed weight if preferred.

```
75 # from Dacier_1996
76 # .0002 is quasi-instantanious
77 # .02 is 1 hour
78 # .2 1 day
79 # 1 is 1 week
80 # 5 is one month
81 # 50 is one year
82 # we invert so scores reflect
83 # mean transition rates (lambda)
84 time_dict = {(0, 1.6): 1. / .0002,
85             (1.6, 3.3): 1. / .02,
86             ( 3.3, 5): 1. / .2,
87             (5, 6.6): 1. / 1.,
```

```
88              (6.6, 8.3): 1. / 5.,
89              (8.3, 10): 1. / 50., }
```

Listing 3.9: Map CVSS to MTTF

As shown in Figures 3.10a and 3.10b, we can also map scored graphs between systems. Listing 3.9 demonstrates how we can map an interval, in this case CVSS score ranges, to a fixed value like the MTTF arrival rates provided by Dacier[107]. In Figure 3.10c we apply a globally fixed CVSS score to the graph which preserves the MTTF mapping, and subsequently remove the mapping in Figure 3.10d to return to the CVSS scoring strategy. By instrumenting the transformation pipeline to support fine grained variations of the structures and values passed to security metrics, we can study how these metrics perform in a variety of scenarios, validate the metrics against a set of functional criteria, and select appropriate security metrics based on performance against a desired reference set.

3.3.2. Security Metrics Catalog

Our architecture for implementing security metrics is straight forward. We declare a *base security metric* type from which all metrics inherit three capabilities. Check Prerequisites: is invoked either directly by the caller or in the metric's own calculate method to ensure all items necessary for the calculation are present. Calculate: returns the resulting measurement. Get Metadata: returns the environment and ancillary data used during the calculation.

This design allows us to implement a library of security metrics with a standard, stateless interface. We have been largely focused on attack graph based metrics, but tests against metrics using other structures (eg attack nets and economic/game theoretic models) suggests this architecture will support any type of security metric we've come across. Listing 3.10 shows the runtime flags available to our library implementation. Because the derived system facts may change based on the rules asserted, after executing a test we can dump all primitive and derived facts to a JSON structure and use that to recreate the environment that produced a specific sample value. Because the original system is seldom referenced in attack graph metrics, we feel this is an important requirement not only for validating existing metrics, but

also in developing more comprehensive, robust security measurements in the future. That is to say, there is a survivor-ship bias in attack graph based metrics, where the only systems that produce an attack graph are necessarily vulnerable between the selected start and end points. If a metric is exclusively operating on an attack graph for security measurement, then we have already determined the system is insecure and exploitable, and are only estimating the extent to which that system subgraph is resilient or susceptible given the current set of facts and interaction rules.

```
secmet.metrics:
  --input_model_name: use this mulval model
  --secmet_ag_name: use this attack graph
  --secmet_ag_path: path to find attack graphs
  --secmet_fg_dot: fg dot file path (overrides path/name)
  --secmet_fg_name: use this fact graph
  --secmet_fg_path: path to find fact graphs
  --secmet_fix_cvss_score: Applies this cvss score to all vulnerabilities.
    (a number)
  --secmet_map_scores: Map AG scores to another domain
  --secmet_model_size: use exammpe models of this size
    (default: 'small')
  --secmet_model_type: use exaple models of this type
    (default: 'enterprise')
  --[no]secmet_plot_intermediate_graphs: Writes graphs to file when true.
    (default: 'false')
  --[no]secmet_random_cvss_score: Applies random cvss score to all
    vulnerabilities.
    (default: 'false')
  --secmet_random_seed: Use this seed for randoms
  --secmet_score_dict: use this score dictionary
  --secmet_score_strategy: Apply this weighting and scoring strategy
```

Listing 3.10: SecMet CLI Flags

In order to validate a metric we can observe its behaviour over a systematically con-

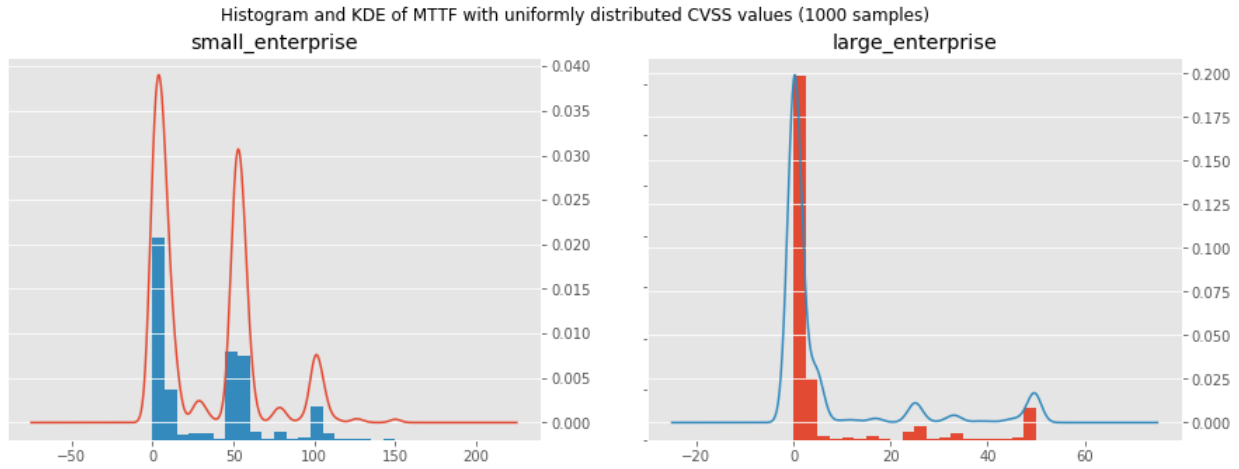


Figure 3.11: MTTF distributions over 1000 random CVSS score assignments (uniform-dist)[108]

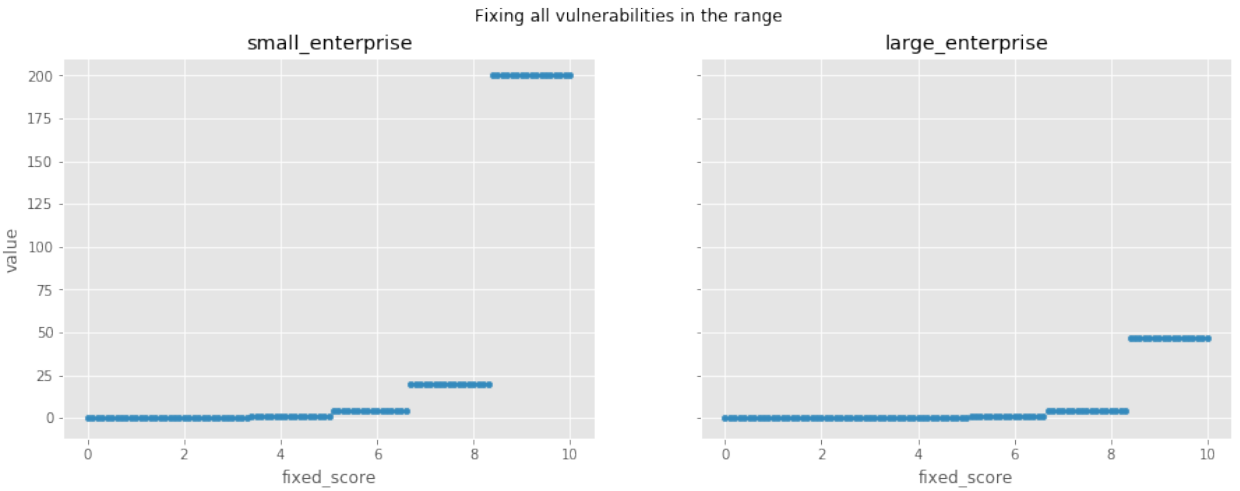
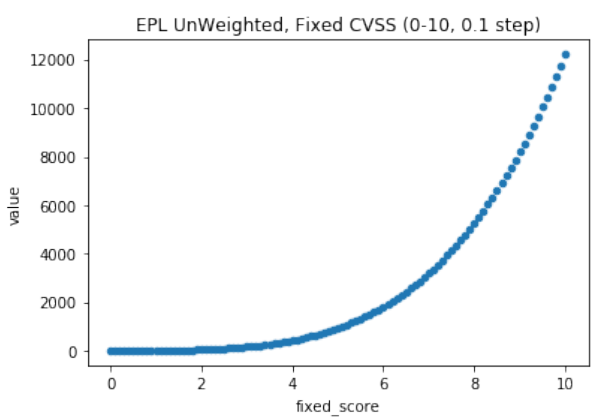


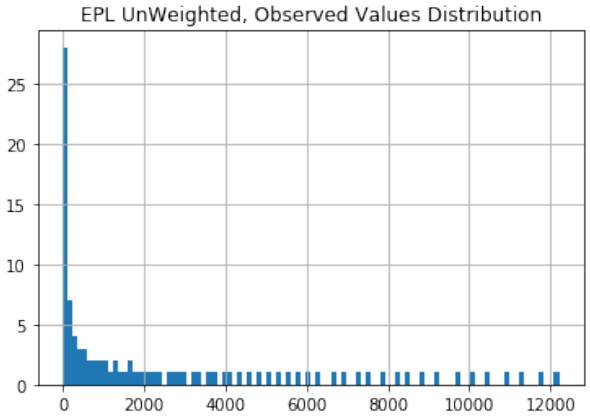
Figure 3.12: Fixing all vulnerability scores to determine lower and upper bounds for metric

trolled set of inputs and confirm it conforms to our predefined validation requirements. We have proposed several validation requirements in Section 3.1.5, although these are likely not appropriate for all metrics or use scenarios. Rather than be prescriptive about specific validation requirements, we only attempt to provide an example of how this framework can be used to characterize the behaviour of security metrics in this work, and leave it up to system stakeholders to determine which characteristics are desirable.

In Figure 3.11 we fix the vulnerability placement of two different input models representing a *small* and *large* enterprises, and select CVSS scores from a uniform random distribution.

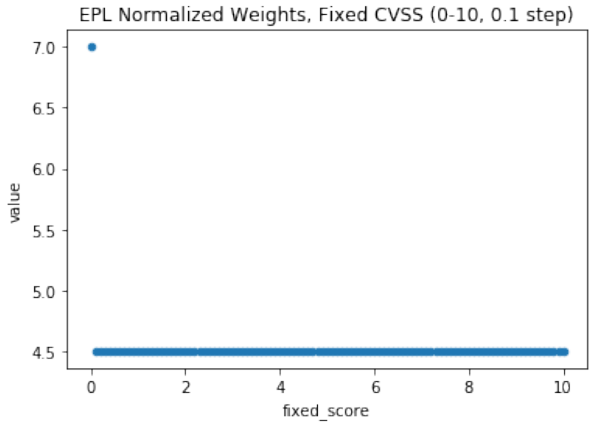


(a) Distribution of observed non-normalized EPL

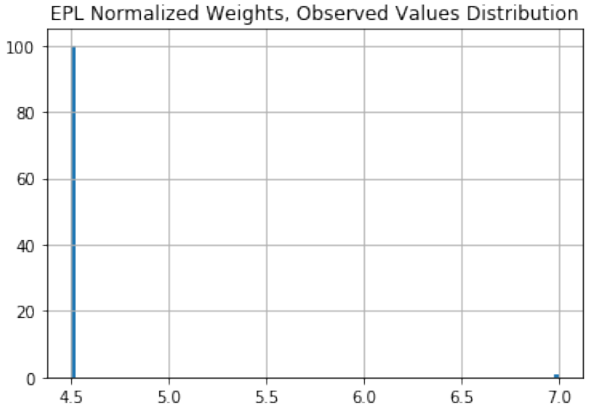


(b) Exponential Distribution of EPL scores without normalization

Figure 3.13: Non-Normalized CVSS-weighted EPL (0-10, 0.1steps)



(a) Normalizing counters the effect of CVSS scoring



(b) All observed values nearly identical

Figure 3.14: Normalized CVSS-weighted EPL (0-10, 0.1steps)

We create 1000 samples in this manner and observe the distribution and kernel density estimates of MTTF[109] scores for all samples. Note that Dacier uses the arrival rates shown in Listing 3.9 to estimate the time an attacker would need to compromise each vulnerability, and we map the randomly assigned CVSS scores to these arrival rates at uniform intervals. The accumulation points observed in the small network occur at seemingly discrete boundaries (x-axis is shown in weeks, where 0 is approximately instantaneous compromise), while the large network demonstrates a more compressed time to compromise over all observed samples, with nearly all vulnerability score distribution resulting in nearly immediate compromise. This distribution of observed MTTF scores is confirmed in Figure 3.12 where we maintain the same input models and vulnerability placement as shown in Figure 3.11, but fix all CVSS scores in the range 0.0-10.0 increasing the score by 0.1 for each sample. The resulting step-wise graph shows the boundary of each mapping interval used, but more importantly, we can clearly observe the range of values in the small network is between 0 and 200 weeks, while the range of the larger network is restricted between 0 and 50 weeks. This discrepancy in achievable MTTF scores means that the size of the network impacts the range of values observed, which should be understood before selecting this measurement for use in operational environments.

In Figures 3.14 and 3.13 we demonstrate the impact of normalization on security measurement values using a procedure similar to that above. In this case we observe the Expected Path Length[13] metric over fixed CVSS scores in the range 0.0-10.0 incrementing again by 0.1 for each sample. In Figure 3.13 we refrain from normalizing the scores used in EPL calculation and observe an exponential increase in observed values from 0 to around 12000 days as all vulnerability scores are raised from 0.0 (easily exploited) to 10 (most difficult to exploit). In Figure 3.14 we run the same experiment, although this time with the following normalization method to determine the transition probability: $p(i,j) = \frac{e_j}{\sum_{k=1}^{n} e_k}$ where n is the number of outgoing edges from node i and e_j is the CVSS exploitability score for the vulnerability in state j. Our immediate observation is the this normalization effectively undermines the effect of CVSS weighting, as the range of EPL values remains nearly constant after perform-

ing this operation. We would also note here that there are several proposed security metrics in the literature that apply some type of normalization to their transition scores with the intent of creating stochastic models, and we are currently evaluating the effects of different scoring and weighting strategies to the behaviour of the underlying metrics.

In figure 4.2 we demonstrate how our implementation can be used to profile metrics and capture performance related metrics of the security metrics under evaluation. Possibly of less concern than the behavioural analyses shown above, when we wish to compare security metric implementations which can be either analytical (for example, the authors propose matrix inversion in their calculation) or stochastic (where simulations are run to estimate the same metric) then we can not only use our framework to confirm both metrics converge on the same value, but also determine which metric does so more efficiently. We therefore enable capturing timings related to each step in the processing pipeline, from input modeling and transformation, to calculation and teardown.

Chapter 4

Security Benchmarking

The number of security metrics available in the literature is somewhat overwhelming. Surveys of security metrics from many[60, 164, 166, 209, 215, 309, 327, 342, 351, 401, 415, 419] sources and perspectives have been conducted, resulting in a multitude of taxonomies for each declaring when and where and how a metric should be applied. What is notably lacking from these surveys, and indeed from much of the literature reviewed, is any empirical evaluation of the values measured by a given metric. The study of security metrics lacks the context needed to support adoption. What we provide in this chapter is a mechanism to establish the needed context by answering how well do these metrics perform individually across a variety of scenarios, and how do they compare with each other for a given model. To build context for security metrics, we break up our experiments into two parts.

The first part *sizes* the metric, examining how it behaves across different types and scales of input networks. We apply the metrics implemented in Section 3.2 to a set of models representative of different deployment scenarios. These models include enterprise networks and core networks at scales we label small, medium, and large. The models act as a reference set against which we can evaluate properties of the implemented metrics. Our primary questions are how do these metrics perform as the size of the system changes, and how do they perform in different types of systems. As our reference set grows, we can develop an understanding of the fundamental or universal security properties we can measure, along with the best metrics for a specific situation.

The second part *ranges* the metric, examining how it behaves as the security of the system under test changes. We select a system model from our standard set and generate an attack model for a scenario. The range of transition values in the attack model varies by metric, but we can, in the general case, fix these values to present a minimally and maximally secure

model with respect to the chosen metric. This bounds the security of a system to a specific range, and measurements within this interval characterize the behaviour of the metric for a scenario. We make observations about monotonicity and sensitivity with respect to a metric and describe properties common across metrics.

We consider ranging and sizing as two aspects of security metric benchmarking. By benchmarking security metrics we validate their fitness for use in general scenarios, and create a frame of reference against which we can measure future security metrics. The rest of this chapter describes the design, testing, and findings from this research.

4.1. Background

When system performance is described, it tends to be in quantifiable terms. The metrics chosen are well understood characteristics like network latency and disk IOPS, the measurement tools are capable of sampling the desired metrics repeatably, and the results are reported with consistency. Benchmarks can be used to compare competing alternatives against a desired metric, to evaluate the effect of changes made to an existing system, or to confirm that expected service level agreements are met. It's easy to draw a line between system performance and return on investment.

In contrast, system security discussions can be somewhat less intuitive. Common measures include compliance level, patch cycle frequency, or the number and severity of vulnerability scan findings. The CVSS[262] framework provides a method to score software vulnerabilities on metrics like impact or complexity, along with knobs to tune the base score according to temporal or environmental factors. Scored vulnerabilities offer a ranking method for remediation priority, but don't give insight into the system's overall security posture since each vulnerability is scored in isolation.

Each vulnerability requires a set of preconditions to exist in order to perform an exploit successfully; if an unpatched service is running but an attacker can't connect to the listening port due to firewall rules, then the vulnerability is not reachable. When a vulnerability is exploited successfully, the resulting postconditions (escalated privileges for example) are

applied to the attacker and environment, potentially opening the way to previously unreachable vulnerabilities. A common representation of the reachability between vulnerabilities in a system is a directed graph, where an edge exists between two nodes when all preconditions needed to compromise the successor are met by the predecessor. By assigning an attacker an initial set of preconditions and a set of target postconditions as the goal, then a sequence of nodes with edges connecting the initial position and the target forms an attack path, and the enumeration of all possible attack paths is the attack graph. The attack graph augmented with CVSS scores forms the basis of many of the model based security metrics described in [309][327][415] which we briefly review in Section 2.2.

There are numerous security metrics published, but what we have encountered in practice supports Verendel's conclusions in [415] and Pendleton's in [309] - that there is a lack of validation for many of the quantitative methods described in the literature. Attack graph based security metrics tend to be studied in isolation, with the result produced being a mathematical derivation demonstrating analytical soundness, possibly accompanied by a simple use-case to illustrate applicability. What is missing from these studies is a standard methodology and data set to verify security performance against.

What we propose is a framework for measuring the relative performance of model based security metrics. Benchmarks exist for most other aspects of computing; for network, compute, and storage hardware, for machine learning, stream processing, cache serving, code compiling, and even for other areas of security like cryptographic libraries, spam filtering, and intrusion detection systems. Paxson [306] provides generally applicable characteristics of *good* benchmarks:

- *Precision* is a limitation of the tool's ability to measure beyond a certain level of detail.
- *Accuracy* errors are differences between the measurement we took and the actual value of the thing we measured. Paxson's example is tcpdump silently dropping packets, leading to difference between measured packet count and actual packets sent.
- *Misconception* errors are differences in what we intended to measure and what we actually measured. Several examples related to incorrect implementation of network

tests (packet loss by retrans count, throughput without filling xfer size filling send buffer)

- *Calibration* is used to reduce errors in accuracy and misconception - 4 strategies including testing edge cases, consistency checks, synthetic test data, and retest with different methods to validate.
- *Metadata* should be associated with each measurement - can limit precision errors and make data re-usable

These properties must be taken into account when designing a measurement instrument for our metric validation framework. Preventing misconceptions about what we are measuring is of particular importance, as the measurement units of many security metrics are not particularly intuitive.

PerfKit Benchmarker is an open source tool originally created at Google that allows users to easily run benchmarks on various cloud providers without having to manually set up the infrastructure required for those benchmarks. PerfKit Benchmarker follows the 5 step process to automate each benchmark run. The Configuration phase processes command line flags, configuration files, and benchmark defaults to establish the final specification used for the run. The Provisioning phase creates the networks, subnets, firewalls and firewall rules, virtual machines, drives, and other cloud resources required to run the test. Benchmark binaries and dependencies like datasets are also loaded in this phase. The Execution phase is responsible for running the benchmarks themselves, and Teardown releases any resources created during the Provision phase. The Publishing phase packages the test results into a format suitable for further analysis such as loading into a reporting system. The metadata returned from the Publishing phase can include verbose details about the actual infrastructure used during the test and timing information for each phase of the run along with the metrics returned from the benchmark itself, providing the level of detail needed to understand the benchmark results in context.

4.2. Methodology

At a high level each attack graph based security metric follows the simplified processing pipeline shown in Fig 3.1.

In Step 1, the inputs include host definitions, accounts and permissions, network connectivity and ACLs, system policies, vulnerability definitions, etc. It is usually assumed these parameters are collected through standard management tools and translated into a common system model for consumption by the attack graph generation engine.

In Step 2, the system model is examined for policy violations or possible exploits that would lead to a given target's compromise. If a compromise is possible, an attack graph is produced. This is the expected input for most of the metrics we have examined, although some also assume the edges have been weighted with CVSS scores first.

The algorithm for computing the metric is run in step 3 with the attack graph as input, and the computed result is returned in the final step.

Our first observation is that survivorship bias[420] is implicit in all AG based security metrics. An attack graph is produced *only* when a system model includes in its definition enough detail to identify possible attacks. An attack graph will **not** be created if a system is totally *secure*. Secure and insecure systems should, in theory at least, be mutually exclusive and collectively exhaustive; unfortunately, an attack graph will also **not** be created if a system is totally *insecure*, but the system model or processing logic is incomplete. So, when validating AG metrics, we must have accepted a priori the selection bias inherent with their use. That is, we are not measuring if a system is secure or not; rather, we are measuring just how insecure that system is.

With this in mind, validation can be seen as a test of how well a metric captures the scale of insecurity which is known to be on the system under test. We propose here a simple methodology for calibrating a metric and gauging it's accuracy in a controlled manner.

What we assume in Algorithm 2 is that the security metric is using some weighting scheme to influence an attacker's selection of paths, which is common in all but the structural metrics (more on these later) we've encountered. So, for CVSS base score weighted AG metrics, we would fix all vulnerabilities at the lowest score (1 in this case) and calculate the metric, and

Require: Valid Attack Graph, N = # partitions
  for n ∈ range (1..N) **do**
    Fix all weights at $\frac{scale}{n}$
    Take measurement
  end for

Algorithm 2: Calibrate Weighted Security Metric

do the same again fixing all vulnerabilities at the highest score for the scale (10 for CVSS). This bounds the range of the metric under test. Creating more than 2 partitions of the score range gives us insight into the behaviour of the metric for this particular attack graph. An example of this calibration is given in 4.5.1

To turn this testing algorithm into a benchmark, we need attack graphs which represent typical deployment scenarios. By far the most common use cases in the literature are small enterprise systems consisting of a limited number of distinct node types (web server, database, firewall, workstation, ...) and a 2-dimensional perimeter. While these examples are easily digestible when describing the applicability of an intensive mathematical derivation, they fall short of validating the new metric as it would be seen in the wild. We propose a standard benchmark set of attack graphs which isolate interesting attack patterns for study. In this way we can target *micro-benchmarks* for specific properties, and integrate or compose models for a more rounded workload examination.

Table 4.1: AG Standard Set - Target Scenarios

	Enterprise	MEC/MANET	Core	Cloud
Small (10-20)	Multi Site	UE Access	Attachment Point	API misuse
Medium (100-200)	Container/K8s	4G/5G	Layer 1&2	Hypervisor
Large (1000-2000)	Insider/Exfil	IoT	SDN/NFV	APT

4.3. Implementation

4.3.1. Benchmark Automation

We refer again to the system performance benchmarking analogy to describe the design considerations and methodology used in the proposed security benchmarking framework. In the performance benchmarking scenario, assume we are considering migrating from a corporate data center to the public cloud, and we want to estimate operating cost for multiple CSPs to base the comparison. Our strategy might begin by characterizing our data center's compute, network, and storage profiles and reproducing them on each of the candidate CSPs. We can size the environments across CSPs by holding the workload parameters constant and tuning each cloud deployment until it matches the expected baseline performance. Once we have established comparable deployment environments between clouds, we can then run the synthetic workload for the same duration on each provider and compare the costs incurred directly. After migration, benchmark tests can be scheduled to ensure SLAs are met and to inform system baseline heuristics and continuous monitoring systems.

If we re-imagine this situation in terms of security metrics, the components needed for validation and verification begin to emerge. In this context, we are still comparing CSPs for cost efficiency, but the constraint is now to maintain the same security posture as our current system.

The first step above was to establish the baseline performance metrics we intend to target on the systems under test. Obviously the target environments would be adjusted to account for differences in requirements - network latency and throughput might deviate based on proximity to the nearest cloud regional data center, or block storage capacity requirements might decrease given the availability of alternate cold storage services.

Similarly, we can take a snapshot of the current security baseline by measuring any or all security metrics for the current state. This involves generating attack graphs and running simulations, or just interrogating the SIEM or other source of aggregated security telemetry for the current patch levels and AV signatures. After establishing the current baseline, we proceed with tuning the remote environments to match the baseline. This might involve scaling instance sizes to align with compute or network objectives, while tuning

for security could lead to rewriting boundary service ACLs to meet the baseline attack surface target. When we tune the candidate environments to match the baseline, we are effectively calibrating our metrics for the current evaluation. After tuning is complete, running the synthetic workload (comprising network traffic generators, endpoint stressers, or disk/CPU/DB/etc relevant operation mixes for example) for a fixed amount of time should yield nearly the same results across all tuned metrics, with the exception being cost, which is the measurement we wanted to take.

To eliminate friction with evaluation and increase likelihood of adoption we have implemented our security benchmarks as an extension of the open source toolkit *PerfKit Benchmarker*[314]. PKB allows users to easily run benchmarks on various cloud providers without having to manually set up the infrastructure required for those benchmarks. PerfKit Benchmarker follows the 5 step process shown in Figure **??** to automate each benchmark run. The Configuration phase processes command line flags, configuration files, and benchmark defaults to establish the final specification used for the run. The Provisioning phase creates the networks, subnets, firewalls and firewall rules, virtual machines, drives, and other cloud resources required to run the test. Benchmark binaries and dependencies like datasets are also loaded in this phase. The Execution phase is responsible for running the benchmarks themselves, and Teardown releases any resources created during the Provision phase. The Publishing phase packages the test results into a format suitable for further analysis such as loading into a reporting system. The metadata returned from the Publishing phase can include verbose details about the actual infrastructure used during the test and timing information for each phase of the run along with the metrics returned from the benchmark itself, providing the level of detail needed to understand the benchmark results in context.

4.4. Monitoring & Alerting

4.5. Results

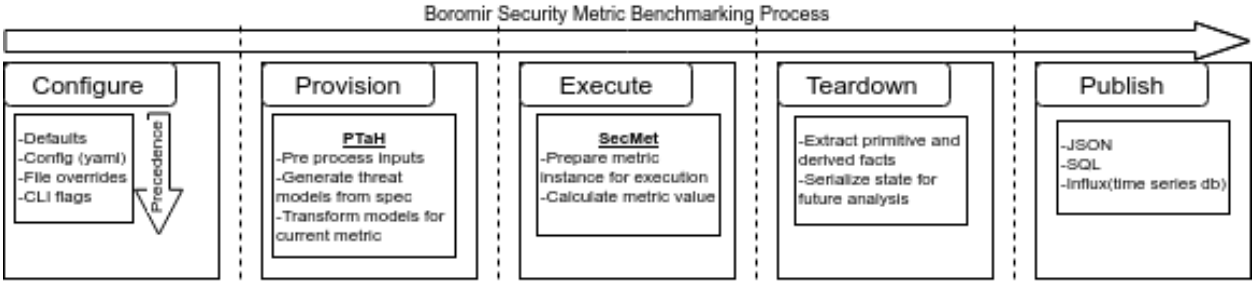
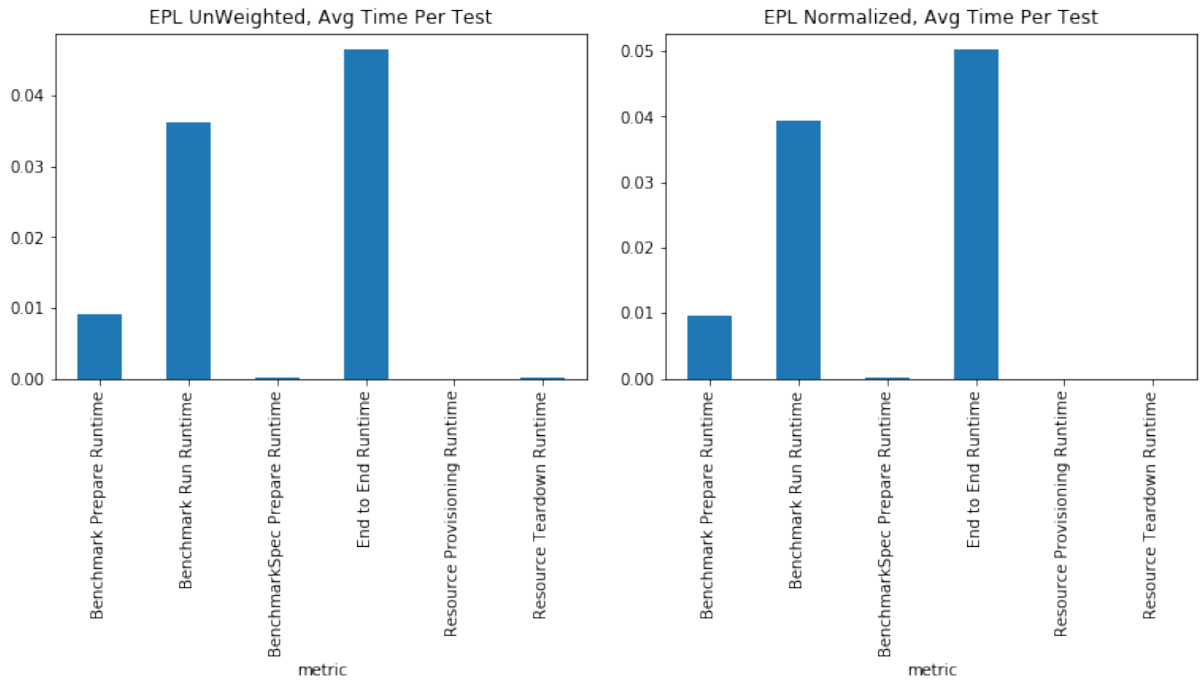


Figure 4.1: Boromir Execution Flow

4.5.1. Benchmark Automation

4.6. Summary



(a) Unnormalized EPL times (b) Normalized EPL times

Figure 4.2: Mean time in each pipeline stage

Chapter 5

Applications of Machine Learning to Security Metrics Research

In this chapter we investigate how machine learning can be used in conjunction with the metrics and measurements we are surfacing to improve our understanding of cyber security.

Specifically, we focus on the following 3 areas in this paper:

1. **Learning valid security metrics**: Can we apply graph clustering and similarity methods to group models by valid cyber security properties?
2. **Model Scoring for Network Design Support**: Can we make use of existing system and threat models, along with associated security metrics, to better model attacker behavior and improve incident response?
3. **Vulnerability Score Fuzzing to predict metric**: Given a network model, can we predict the values of specific security metrics through classification or regression?

In the December 2019 workshop *Implications of Artificial Intelligence for Cybersecurity*[361], one of the key takeaways identified was the need to expand the connections between cyber security and applications of artificial intelligence and machine learning. In this work we have so far focused on making cyber security measurable, focusing on instrumentation for automation and autonomy. Programmatic access to security metrics through automation opens up a wide variety of applications involving, and can itself be improved by, current techniques in machine learning. In this chapter we describe the design details for the experiments we are conducting using the SMaaS environment. While adversarial AI and attacks on ML models are areas of concern today, the direction of this work is *not* in applying SMaaS to evaluate the security of machine learning. Instead, we propose to investigate the use of specific machine learning techniques to improve cyber security through the metrics framework we developed above.

5.1. Background

Machine learning techniques can be broadly grouped into 4 categories based on the types of problems they solve. **Clustering** problems try to find groupings in data sets by minimizing distances from members of the same group and maximizing distances between members of differing groups. **Classification** problems map new instance data onto a discrete set of values representing categories or labels, while **regression** maps new data onto a continuous set of values. **Rule extraction**[121] is a statistical inference method used to predict responses from given input patterns. It is also common to describe machine learning by one of four types of learning used. **Supervised** learning is provided labeled training data to build models from, while **unsupervised** learning is trained on unlabeled data. **Semi-supervised** learning is used with a mix of labeled and unlabeled data. **Reinforcement** learning[395] maximizes values from an internal scoring functions to learn a model about the environment.

The use of machine learning techniques in cyber security is just starting to be explored. The two surveys[68, 439] published in 2016 and 2018 review a wide range of ML and RL methods specific to the area of intrusion detection. In [65] the authors survey the field of networking generally for uses of machine learning. Their findings cover areas in traffic management like routing and congestion control, as well as resource and fault management topics. On the subject of network security, the subject areas reviewed were again narrowly focused on intrusion detection.

In this work we describe methods for evaluating cyber security that incorporate the underlying structure of the (computer) network under test. Graphs are a natural way to represent network connections and the relationships between system components, but preserving that structure adds complexity to the cost of analysis. Modern machine learning techniques make use of optimizations and transformations to reduce their complexity, and graph learning methods are no exception. Similar to preprocessing an image set to uniform dimensions prior to training, many graph learning methods presuppose an embedding of the graph data as input requirement. Goyal's graph embedding survey[153] finds 4 broad categories of these methods:

1. **Node Classification**: Predict the label of a node based on its embedding.
2. **Link Predicton**: Predict edge based on embeddings of nodes it joins.
3. **Clustering**: Community detections and node groupings.
4. **Visualization**: Interpreting more than a few dozen nodes becomes difficult.

Graph embedding can mean either embedding the entire graph in vector space, or embedding each node in vector space. The goal in either case is to encode the graph data onto a lower dimensional space while retaining the relevant structural relationships. Common practice is to learn graph embeddings by defining the encoding and similarity functions, and then optimize the encoding parameters which maximize the similarity. How the similarity distance is measured and which properties are selected for feature encoding are some of the challenges in deciding the best embedding method[153].

In the December 2019 workshop *Implications of Artificial Intelligence for Cybersecurity*[134], one of the key takeaways identified was the need to expand the connections between cyber security and applications of artificial intelligence and machine learning. In this work we have so far focused on making cyber security measurable, focusing on instrumentation for automation and autonomy. Programmatic access to security metrics through automation opens up a wide variety of applications involving, and can itself be improved by, current techniques in machine learning. In this section we describe the design details for the experiments we are developing using the SMaaS environment. While adversarial AI and attacks on ML models are areas of concern today, the direction of this work is *not* in applying SMaaS to evaluate the security of machine learning. Instead, we propose to investigate the use of specific machine learning techniques to improve cyber security through the metrics framework we developed above.

Machine learning techniques can be described by the 4 key types of of problems they solve. **Clustering** problems try to find groupings in data sets by minimizing distances from members of the same group and maximizing distances between members of differing groups. **Classification** problems map new instance data onto a discrete set of values representing categories or labels, while **regression** maps new data onto a continuous set of values. **Rule**

extraction[121] is a statistical inference method used to predict responses from given input patterns.

It is also common to describe machine learning by one of four types of learning used. **Supervised** learning is provided labeled training data to build models from, while **unsupervised** learning is trained on unlabeled data. **Semi-supervised** learning is used with a mix of labeled and unlabeled data. **Reinforcement** learning[395] maximizes values from an internal scoring functions to learn a model about the environment.

Machine learning is a fast moving research area. We consult the 2019 *Deep Learning in Mobile and Wireless Networking: A Survey*, [457], 2018 *A comprehensive survey on machine learning for networking*[65], and the 2018 *Machine Learning and Deep Learning Methods for Cybersecurity*[439] to identify current techniques and canonical literature for our topic.

MulVal Facts : Datalog facts defining system model: (Hosts, Policies, Principals, Vulnerabilities, Interaction Rules) + (Attacker Location, Goal) MulVal Output: DAG of possible paths attacker can reach goal (nxn connectivity matrix)

5.2. Security Metric Prediction

Problem Type: Classification and Regression problem.

There are quite a few scenarios that can be described as 'Metric Prediction'. In general we approach these as supervised categorical classification or regression tasks, where SMaaS is used to create a labeled dataset of network models or attack graphs with the resulting security metric(s). Training and validation proceed as usual, with the resulting model able to predict the metric of new network states or topologies without needing the SMaaS system.

1a. Vulnerability Score Perturbation to predict metric

Goal: Given a weighted transition matrix $\longrightarrow$ predict specific metric

Method: Fuzz vulnerability weights (won't affect AG structure)

- Find best fit function (LR, . . .)
- Compare time/perf over simulation for small/med/large matrixes
- Compare performance for different metrics (within/outside same class)

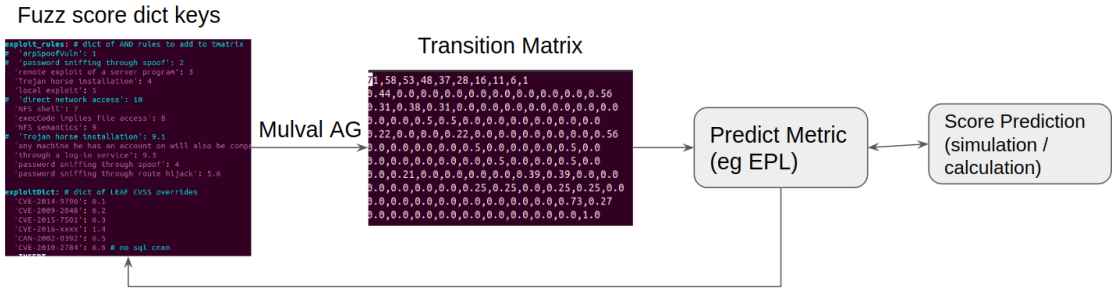


Figure 5.1: Learning Metrics from Vulnerability Score

- Fuzz subset of vulns for PCA or confuse

1b. Vulnerability Placement Fuzzing

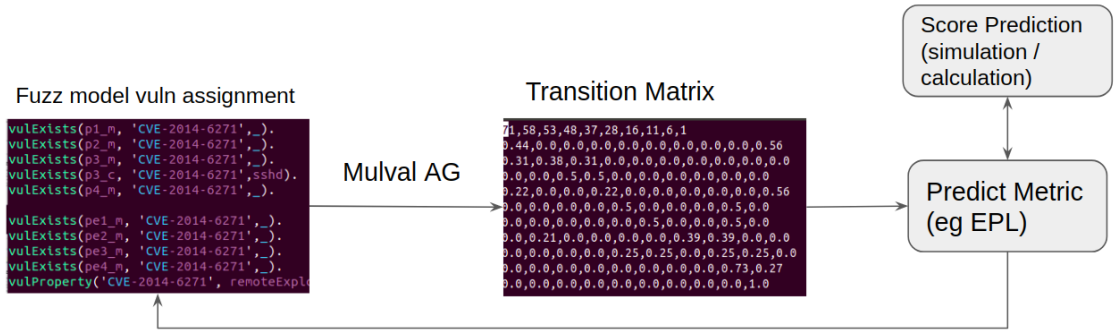


Figure 5.2: Learning Metrics from Vulnerability Placement

Goal: Given a MulVal model $\longrightarrow$ predict specific metric

Method: Fuzz vulnerability placements (will affect AG structure)

- Can we learn if an AG will be produced (connectedness)?
- Can we predict any metrics?
- Might be a good application for GNNs
- Could either learn on tmatrix (easy) or MulVal model (harder)

1c. Network Model Fuzzing

Goal: Given a MulVal model $\longrightarrow$ predict specific metric

Method: Fuzz network facts (will affect AG structure)

- Can we learn if an AG will be produced (connectedness)?
- Can we learn how to make a valid Network model?

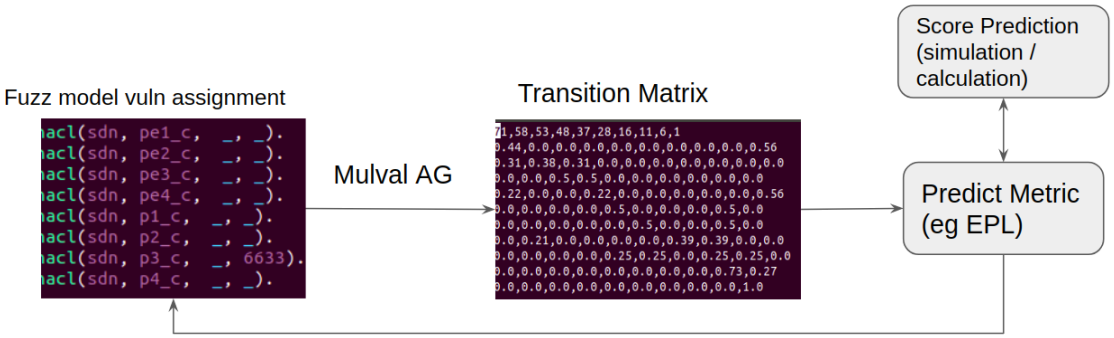


Figure 5.3: Learning Metrics from Network Topology

- Prolog syntax/token understanding needed

5.3. Model Enhancements

The reliability of a measurement depends on the reliability of the input. *Garbage in, garbage out* is especially true for model based calculations. In Section 3.2.1 we discussed the role *interaction rules* play in defining the possible attacks against a system, and that a model missing relevant interaction rules will falsely report no possible attacks when, in fact, they do exist.

In order to ensure complete coverage of IRs for the domains we model, we need to have these rules defined systematically. A good starting point is MITRE's CAPEC[100], the Common Attack Pattern Enumeration and Classification dataset. While this corpus includes primarily application type attacks, another dataset, CyBOX (recently moved into STIX), includes attack patterns observable at lower layers like infrastructure and physical such as those we defined in Section 3.2.1. Recall from Section 2.1 that an interaction rule can be represented as a *Horn Clause* of the form $A \leftarrow B_1, B_2, \ldots, B_n$, where $B_1, B_2, \ldots, B_n$ are facts defining preconditions that must be true to assert A is true. Both CAPEC and CyBOX include preconditions with each attack pattern, but sadly they chose to make this field free text and not categorical. Noel in [289] describes text mining techniques for many scenarios but prerequisite enumeration is not one of them.

`This attack requires the following:`

1. The application uses environment variables.
2. An environment variable exposed to the user is vulnerable to a buffer overflow.
3. The vulnerable environment variable uses untrusted data.
4. Tainted data used in the environment variables is not properly validated.

The example above lists the CAPEC preconditions for the *authentication abuse, ID:114* attack pattern. There are currently 517 unique attack patterns, along with associated pre-requisites and *consequences*, which capture post conditions like privileges gained from a successful attack. Our goal in this work is to expand the knowledge base for threat model generation to more accurately represent known patterns, and to accommodate future patterns as they become known.

5.3.1. Interaction Rule Mining

Problem Type: Association Rule Learning

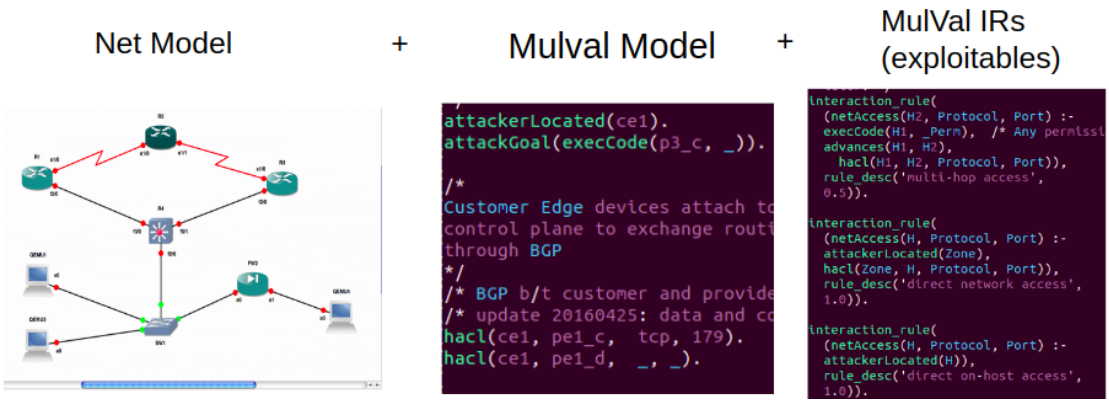


Figure 5.4: Association Rule Learning

Goal: Given a Network model ⟶ learn vulnerability rules and conditions

- Need fine grained data (ports/protocols/services/versions)
- Can we learn conditions for existing exploits?
- Can we learn conditions for new exploits?
- Output needs to comply with datalog AST (need ANTL/Thrift here maybe)

At the heart of MulVal is the interaction rule, which describes how each the set of facts

combine to derive new information about the system. These derived facts then describe how attackers can advance through the system towards a target. Developing interaction rules is currently an artistic endeavor undertaken by a subject matter expert... that is to say, it is error prone, and MulVal includes around 30 interaction rules to derive the following 8 facts:

```
1 derived(accessFile(_machine,_access,_filepath)).
2 derived(accessMaliciousInput(_host, _principal, _program)).
3 derived(canAccessHost(_host)).
4 derived(dos(_host)).
5 derived(execCode(_host, _user)).
6 derived(logInService(_host, _protocol, _port)).
7 derived(netAccess(_machine,_protocol,_port)).
8 derived(principalCompromised(_victim)).
```

Listing 5.1: Mulval Derived Facts

As we have demonstrated previously, the existing MulVal model leaves large gaps in defining the entire attack surface of a network. To begin covering the possible IR set it will be necessary to automate the rule generation process.

- Map existing actively maintained taxonomies like MITRE's CAPEC (common attack patterns), STIX (structured threat information), or MAEC (malware attribute enumerations) to an ontology of simple subject-verb-object relations.
- Translate this ontology into DataLog Horne clauses
- Translate the populated taxonomy information to DataLog Facts
- Use these IR rules to seed the next phase of ML or RL based rule learners and rule refiners [273, 159]
- Evaluate the efficacy of the ML rule learners by comparing to established attack patterns (withhold a subset of MITRE rules for testing, validate new rules not in test set through penetration test)

5.3.2. Agent Based IR Learning

Problem Type: Reinforcement Learning

Goal: Demonstrate the feasibility of using RL to model attacks. Further, identify if and when security metrics are appropriate to incorporate into the environment responses, agent value function, or agent policy to model attacker or defender behaviour.

Reinforcement learning[395] is a computational method of building an optimal set of interactions between an agent and its environment to achieve a specific goal. A *policy* defines the agent's behaviour for a given environment state. A *reward signal* defines the goal as a single reward value returned at each time step. A *value function* is used to predict the maximum reward available to the agent in the given environment. Environment state can be described using a Markov decision process similar to how we have already modeled attacker state in some of our metrics.

Model-free RL can be thought of as trial and error based learning, where the agent doesn't need to understand how its actions affect the environment. In model-free RL, Q-learning is the most well studied and widely used method[65].

Model-based RL allows an agent to make inferences about how the environment will behave by planning how possible actions will change the environment's state. The simple tic-tac-toe example given by [395] uses the 3x3 board as a model which can be used by an agent to anticipate the results of potential moves and plan for an optimal strategy against an opponent.

In our case, we already have two distinct models for this type of reinforcement learning. The first model is the 'normal use' transition matrix of the system model that shows connectivity between elements given as a set of user and system principals and permissions, network ports and protocols, and access control lists. The second model is the transition matrix of the exploitable paths in the resulting attack graph.

5.4. Methodology

As discussed in the literature review, the majority of machine learning applications in the cyber security realm have been focused on intrusion detection techniques. Rather than revisit those efforts, through our previous work we are in a unique position to take advantage

of existing measures of security to assist in training and optimization of machine learning algorithms. While there are many security metrics in the literature, we focus primarily on those that already rely on attack graphs to calculate a metric. Pendleton's survey[309] focuses on metrics that quantify attack and defense interactions, and these are the broadly descriptive measurements that would provide the most benefit to operators in the field. These metrics are classified as measuring one or more of Vulnerabilities, Threats, Defenses, or Situations. Situations in this case is a comprehensive metric, with Pendleton's example subgroups measuring security state over time, successful attacks over time (incident rate), and return on economic investment. Again we find a subjectivity in the assignment of users' susceptibility and attack and defense effectiveness scores. Even CVSS is bound to variance for any vulnerabilities not pre assigned a score. In comparison, analogous system wide performance metrics like those found in workload simulations (YCSB for example) will be fairly deterministic. Verendel[415] makes a critical analysis of the claim that security is quantifiable. The premise is that most of the published models and metrics that attempt to measure security lack the scientific rigor to corroborate or validate their hypothesis. The scope of is limited to operational security measurements and assumes measurement primitives include systems, threats, and vulnerabilities.

Our methodology then is to explore areas of machine learning that have have not yet been available to the cyber security community. To this end, we have a variety of options available, not all of which will be productive. Our first approach is to label a set of system models with the value assigned by respective security metrics. In this way we create a labeled training set that can be used to train a model which avoids the expensive overhead of evaluating metrics continuously through analytical methods. Given the nature of many stochastic metrics, there is also a great potential for applying unsupervised learning methods to our data set, which we will also describe in the next sections.

5.4.1. Data Generation and Representation

Recall that Boromir drives the security metric benchmarking process and manages sample

persistence. The following command generate 1000 samples of the standard set metrics for the default models defined in the environment. Setting random scores is optional, but makes non-stochastic measurements a little more interesting.

```
> ./run_boromir.py --boromir_run_count=1000 --secmet_random_cvss_scores
```

When a test run kicks off, Boromir relays the user inputs and security metric specifications to PTAH to provision the system models which will be used for the test.

All facts used to build the models are conveniently collocated in XSB with the MulVal[299] attack graph engine. This allows us to solicit interaction rules from users or other processes while the models are being assembled. When a run is over, the fact base is dumped from XSB and packaged with the test's metadata, so every sample contains the information needed to recreate the environment it was taken from.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
sample_uri    1000 non-null object
reduced_ag    1000 non-null object
orig_ag       1000 non-null object
fact_graph    1000 non-null object
value         1000 non-null float64
dtypes: float64(1), object(4)
memory usage: 39.2+ KB
```

Listing 5.4.1 shows a dataframe holding 1000 samples from a boromir run. The *sample_uri* field is a unique uuid for each row, and *value* is the result of the metric calculation. The remaining three fields hold the fact graph, the attack graph, and the adjacency matrix equivalent subgraph of the attack graph. We describe the reduction processes provided by this framework in Section 3.2.3, so here we simply refer to the networkx inherited function to dump the graph as a sparse matrix.

	sample_uri	w_0_0	w_0_1	w_0_2	w_0_3	w_0_4
0	f3c1b87e-cad1-431f-ac63-eac72b694c84	0.0	5.49	0.0	0.0	0.0
1	0514df06-4242-451a-b737-41057c26dc00	0.0	7.74	0.0	0.0	0.0
2	a6c94818-2807-484a-83c5-d2f227a37a45	0.0	6.21	0.0	0.0	0.0
3	8b265a5c-f662-42fd-a8a1-f084ef4a66e1	0.0	2.05	0.0	0.0	0.0
4	9dc8f994-84a6-40d4-ae6d-bf1cfdba007d	0.0	7.00	0.0	0.0	0.0

	w_0_5	w_1_0	w_1_1	w_1_2	{\ldots}	w_4_3	w_4_4	w_4_5	w_5_0	w_5_1	w_5_2
0	0.0	0.0	0.0	5.81	{\ldots}	0.0	0.0	5.07	0.0	0.0	0.0
1	0.0	0.0	0.0	5.21	{\ldots}	0.0	0.0	6.43	0.0	0.0	0.0
2	0.0	0.0	0.0	1.85	{\ldots}	0.0	0.0	8.10	0.0	0.0	0.0
3	0.0	0.0	0.0	5.71	{\ldots}	0.0	0.0	4.82	0.0	0.0	0.0
4	0.0	0.0	0.0	9.67	{\ldots}	0.0	0.0	7.84	0.0	0.0	0.0

	w_5_3	w_5_4	w_5_5	score
0	0.0	0.0	0.0	4.00
1	0.0	0.0	0.0	12.00
2	0.0	0.0	0.0	16.00
3	0.0	0.0	0.0	0.75
4	0.0	0.0	0.0	40.50

[5 rows x 38 columns]

5.5. Results

In Figure 5.5 we show the error rates where 5.5l is the histgram of securty values dis-

tributed over the label space.

For a graph $G = (V, E)$, a node embedding function $f : u \rightarrow \mathbb{R}^n$ maps each node $u \in G$ onto a d dimensional set of real values called a feature vector. For any two nodes $u, v \in G$, a similarity function $sim(u, v)$ measures the strength of the relationship between the two nodes. Because nodes can be related in many different ways, there are multiple ways to measure similarity. The proximity of the encoded nodes in the embedding space reflects the similarity of the nodes in the original graph. Current embedding approaches include matrix factorization[46][18], random walks[311][155] and deep learning[421][205]

[216]

[161]

5.6. Summary

In the literature we reviewed there were over 500 distinct security metrics identified. The surveys each provided their own classification systems which were appropriate for the analysis they conducted, but none of these taxonomies generalize well to classify all types of security metrics. We describe properties common to all metrics, identify overlaps in the various taxonomies, identify points of confusion between existing metric hierarchies, and describe a suitable and intuitive system for classifying any current or future security metric. In using the CyBOK as the underlying classification system we are also able to determine the distribution of metrics in each topic and identify areas of limited coverage which would benefit from future research.

In reproducing the results from the literature, we faced several issues during implementation, particularly with model based security metrics. Often assumptions were made about the intermediate processing of the pipeline that weren't surfaced in the supporting examples of the publication. We note that many of the survey authors describe security metric validation as an area of concern in security metric research. In response to these concerns and to move forward in our own research, we identified a generalized four step processing pipeline that separates the core steps of this process. By following this workflow we have identified

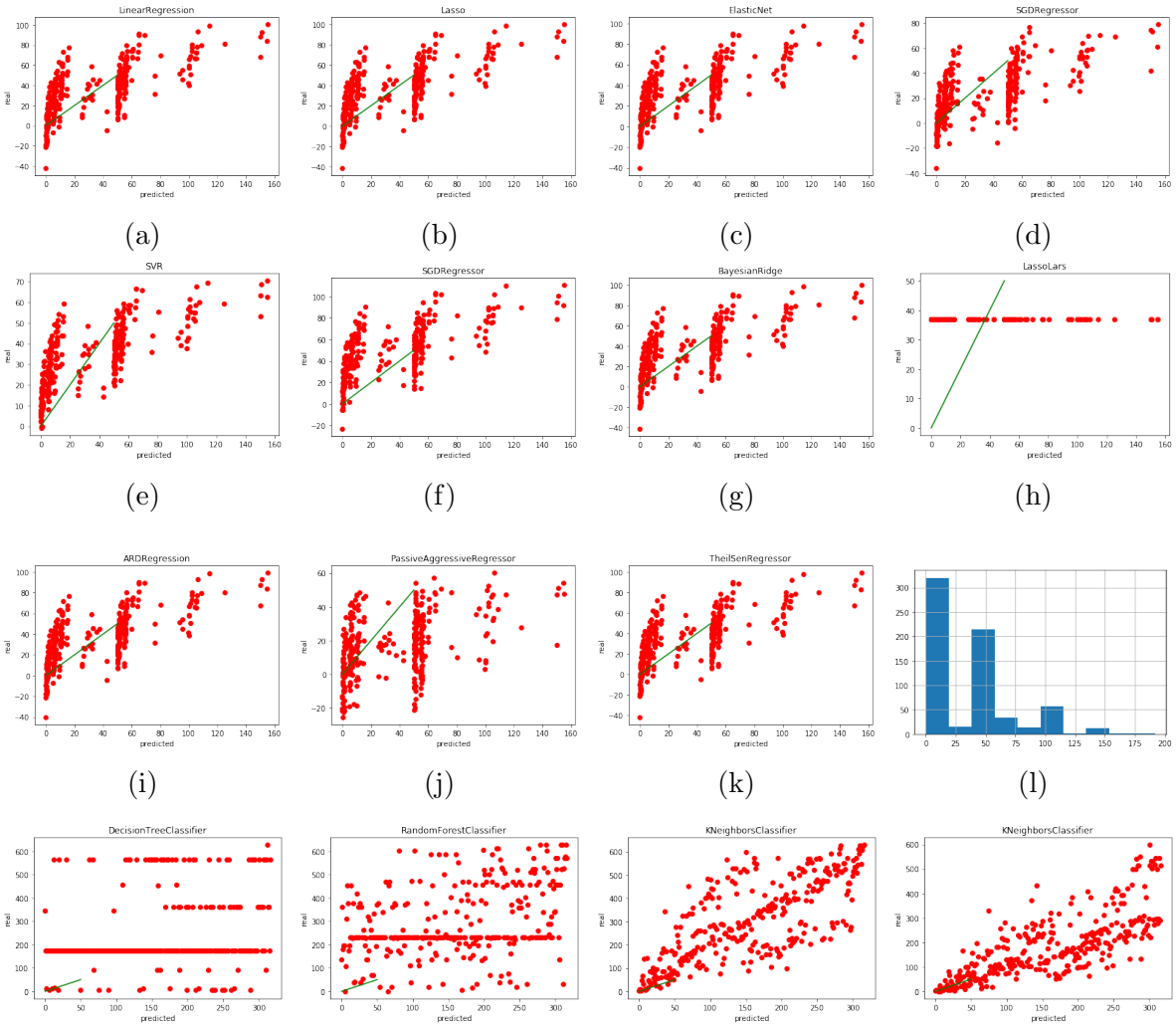


Figure 5.5: Classifier Error Plots (Predicted vs Actual) for the system's calculated security

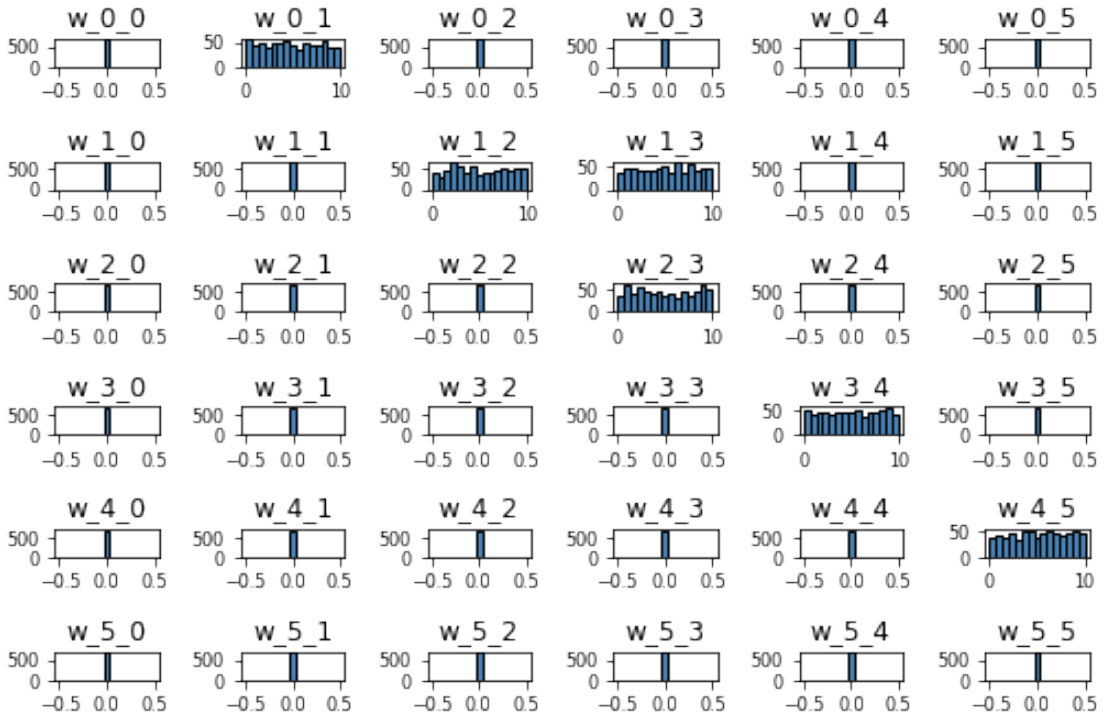


Figure 5.6: Histograms of observed value distributions for each feature

and implemented many reusable preprocessing components, allowing us to rapidly add new security metrics from the literature and immediately test the performance of those metrics against a growing number of input models we use as our reference set.

By enforcing the 4 stage pipeline abstraction, we achieve several benefits. Each phase is modular so that replacing any piece in the pipeline is straight forward. By plugging in the static reference set to the input phase we create a *unit test* of sorts for our metric library, SecMet. With PTaH, the preprocessing and transformation handlers described above, we can articulate how any or all of the security metrics will behave under a variety of conditions for any given input - not just the reference set we describe above. This, in theory at least, should make characterizing the behaviours of security metrics on internal or sensitive systems as simple as adding input adapters to the existing set, which already includes Nessus, OVAL, NVD, CVE, and now SSFNet. By replacing our validation PTaH with whatever workflow execution engine is already in place, Apache Beam or Storm for example, the SecMet catalog becomes a drop in security measurement aid to support SecDevOps which we refer to as

Security Metrics as a Service (S-MaaS).

Our takeaways from the experiments described above indicate that, while validating security metrics is not done rigorously in many of the publications, a mechanism for validation and analysis is not out of reach. By streamlining the development and evaluation process with automation, we aim to lower the barrier to entry in the field and allow researchers to spend more time developing and analyzing security metrics, which in turn should encourage more secure systems being deployed.

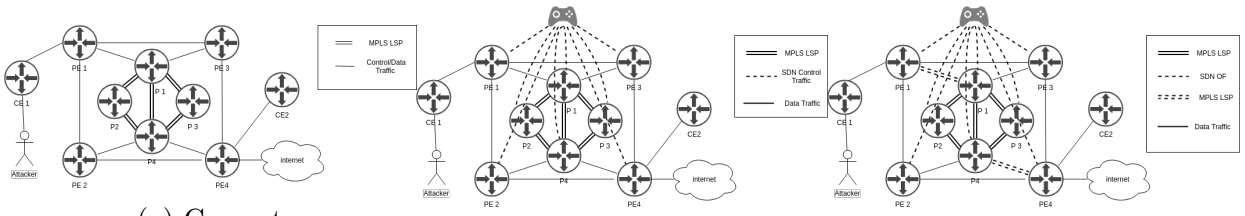
Chapter 6

Case Study

6.1. Carrier Network Migration

We consider the use case of a network operator migrating core infrastructure from traditional switching and routing elements to a centrally managed SDN architecture. We assume the migration process will occur in three discrete phases, with a single intermediate migration state.

6.1.1. Migration Path



(a) Current (b) Transition (c) Future

Figure 6.1: Network migration models from current to future

The primary network elements we consider for this analysis are *Customer Edge (CE)* routers, *Provider Edge (PE)* routers, and *Provider Core (P)* routers. *CE* routers attach to *PE* routers and communicate information about the customer network topology to the provider. *PE* routers exchange routing information with other *PE* routers about the networks they are attached to. *PE* routers also learn from *P* routers what paths are available through the provider network. Once routing and signalling information is established, the *PE* and *P* routers can forward customer traffic through the provider network. In conventional networks routing and signalling information is exchanged directly between adjacent routers, whereas the controller receives this information in SDN architectures. To determine

the correct path through the network, routers and switches are required to parse frame and packet headers to identify the source and destination of incoming traffic. *Multi Protocol Label Switching (MPLS)*[37] is used to reduce the overhead of packet processing and isolate customer traffic by applying labels on incoming packets. Labels speed up routing through core and transit networks by reducing the effort needed to process the header at each hop. We have reduced the scale and scope of the network models to capture some of the key changes in the architectures during migration while reducing overall complexity. These representative models allow us to isolate the impact of specific architecture changes on the system's security.

The current network model depicted in Figure 6.1a captures the existing architecture elements such as distributed routing protocols and hardware and operating system level components. When customer traffic enters the Ingress Router PE1 from the customer edge CE1, access control lists are enforced to drop any packets with a destination address matching the address of the core infrastructure. Traffic flows that are not denied by the ACL are then routed through the MPLS core and forwarded to the appropriate Egress Router. In practice, the number of ACL rules maintained on each Ingress Router could exceed 100,000 entries.

The transition state network in Figure 6.1b retains the same logical connectivity as the current network by leveraging ACLs to restrict customer traffic. This model introduces a global SDN controller along with the supporting infrastructure to facilitate a centrally managed SDN environment. Proprietary switching hardware has been replaced by merchant silicon and the vendor specific applications that control the hardware have been abstracted and moved onto the hypervisor. The result is that, while ACLs can now be centrally managed, the attack surface of the network has increased with the addition of the SDN components.

The final network model in Figure 6.1c assumes the same underlying infrastructure as the transitional SDN model in 6.1b but places the user traffic bound for the internet in an MPLS VPN tunnel[281][336] instead of the default global routing context. This allows us to study the overall effect on security of isolating the customer traffic flows and preventing the core elements from being directly addressable.

Table 6.1: Network Elements

Device	Version	Function
Cisco 12000 series	IOS 12.0(32)S11v	Provider Edge
Cisco ASR9000 series	IOS XR Version 4.3.1	Provider Edge
Cisco CRS1	IOS XR Version 4.3.3	Provider Edge
Juniper T-series	Junos 12.3R3-S4.10	Provider Edge
Cisco CRS1	IOS XR Version 4.2.4	Core
Juniper M320	Junos 13.2R2-S5.2	Route Reflector
Merchant Silicon switches, routers, OLTs	Open Network Linux	Fabric
SDN Controller (local)	Juniper Contrail	SDN
SDN Controller (global)	ECOMP	SDN
SDN Controller OS	Ubuntu 14.04	Application Host
Network Function Virtualization Host	RHEV 2.2/ KVM 83	Hypervisor (PE/P/RR/TE)
Merchant Silicon switches, routers, OLTs	Open Network Linux	Fabric

Some information describing the components of the three architectures is given in Table 6.1, while the ports, protocols, and services listing in Table 6.2 provides details on current and target state network services. How services are accessed across boundaries, what vulnerabilities are present, and how data flow shifts when moving from decentralized to centralized control are the key elements in this analysis. These details are translated into the MulVal input model described in Section 3.

6.1.2. Vulnerabilities

The Common Vulnerability Scoring System[262] (CVSS) is an open framework used throughout government and industry to report the severity of specific security vulnerabilities. CVSS scores range from 0 to 10 based on the vulnerability's exploitability and impact, with a score of 10 signifying the highest severity. Exploitability is calculated by determining the access vector, access complexity, and number of authentication attempts required to exploit the vulnerability, with higher exploitability values equating to an easier compromise. Impact scores are determined by identifying the scope of a successful exploit on the vulner-

Table 6.2: Ports, Protocols, and Services

Protocol	Port	Service	Boundary
IGP	-	OSPF	P
TCP	179	BGP	PE↔ PE, PE↔ CE
TCP/UDP	363,1698,1699	RSVP	P
PIM	-	-	P
Telnet	-	Telnet	
TCP	22	SSH	
UDP	161,162	SNMP	
UDP	123	NTP	
IGP	-	OSPF	P ↔ SDN local
TCP/UDP	646	LDP	P ↔ SDN local
TCP	179	BGP	PE↔ CE and P/PE↔ SDN local
TCP/UDP	363,1698,1699	RSVP	P ↔ SDN local
PIM	-	-	P↔ SDN local
Telnet	-	Telnet	
TCP	22	SSH	
UDP	161,162	SNMP	
SAA/TWAMP	-	-	
UDP	123	NTP	
TCP	6633	OpenFlow	SDN Global ↔ SDN local

able system's confidentiality, integrity, and availability. CVSS scores used in this research were queried using a local copy of the National Vulnerability Database (NVD) synchronized via MulVal's built-in mechanism and augmented with scores provided by vendors when an official Common Vulnerability Enumeration (CVE) designation was not available.

MulVal was originally designed with enterprise network security in scope, but recent research[17, 40, 168] has provided extensions that allow for modelling of individual network infrastructure attacks. In 2019 [384] presented a coherent set of facts and rules for modeling Layer 1-3 attacks in communication networks which provides the needed semantics to define the threats posed in Table 2.1.

The network vulnerabilities listed in Table 6.3 have been identified to represent the types of attacks within the scope of this project. Our intent is to identify which types of attacks are mitigated by moving to SDN and what unplanned attacks are introduced. To aid in this analysis we add potential vulnerabilities (eg, ACL misconfigurations, 0-days, negligent admins, etc. . .) by assigning a theoretical CVSS score to the vulnerability and adding it to the network model.

Table 6.3: Vulnerabilities

CVE ID	Vulnerability Description	Affected Hosts
CVE-2012-1342	ACL Bypass (privilege escalation)	IOS 12.0
CVE-2011-4012	ACL Bypass (privilege escalation)	IOS 12.0
CVE-2011-2395	Bypass (privilege escalation)	IOS 12.0
CVE-2010-4685	Bypass (privilege escalation)	IOS 12.0
CVE-2007-5381	BoF (remote code execution)	IOS 12.0
CVE-2007-4295	Malformed Packet (remote code execution)	IOS 12.0
CVE-2015-0694	NACL Bypass (privilege escalation)	IOS XR
CVE-2014-3396	ACL Bypass (privilege escalation)	IOS XR
CVE-2013-3464	BoF (remote code execution)	IOS XR
CVE-2013-1234	BoF/DoS (remote code execution)	IOS XR
CVE-2014-6379	RADIUS Bypass (privilege escalation)	JunOS
CVE-2014-3818	BoF (remote code execution)	JunOS
CVE-2014-3816	(privilege escalation -\textgreater authenticated user)	JunOS
CVE-2013-6618	Remote code execution	JunOS
CVE-2015-7501	ODL remote code execution	OpenDaylight
CVE-2015-4000	ODL MitM (priv escalation/remote code exec)	OpenDaylight
CVE-2015-1778	ODL Auth Bypass (priv esc/remote code exec)	OpenDaylight
USN-2949-1	use-after-free vulnerability in the Linuxkernels CXGB3 driver(DoS, remote code execution)	Ubuntu 14.0.4
CVE-2014-9769	PCRE regex (DoS, remote code execution)	Ubuntu 14.0.4
CVE-2010-2784	RHEV/KVM local priv escalation, DoS	RHEV 2.2/KVM 83
CVE-2014-6271/7169	DoS/remote code execution (ShellShock)	Bash 4.3

After reviewing known CVE's to identify attack vectors, we apply hypothetical vulnerabilities to the proposed networks which affect the newly introduced architecture components. Table 6.4 lists examples which would reflect the threats identified during migration.

In addition to network resources and connectivity attributes, vulnerability data is also assigned to each host. Vulnerability information has the form **vulProperty(vulnID, accessType, effect)** and can be assigned to one or more hosts **vulExists(host, vulnID, program)**, where services running on a host are defined as **networkService(host, program, protocol, port, userPriv)**

Example vulnerability definitions for the network elements listed can be found in Table 6.4. For this analysis we assigned the same vulnerabilities to the Transition and Final State network elements and only altered the connections between hosts and the related protocols as specified in the PPS listings above. Within our preliminary experiment parameters the result was that *no attack path could be found between the attacker and the target.* While this finding is in itself interesting, to facilitate analysis we have introduced theoretical vulnerabilities into the Current, Transition, and Final network models as described below to demonstrate the end-to-end CSAF flow.

Table 6.4: Hypothetical Vulnerabilities

Vulnerability Class	Examples	Possible Effect	Exploitability	Impact
ACL Bypass	Misconfigured ACLs on PE devices could allow an attacker to send packets addressed to core network elements.	Privilege escalation Remote Code Execution	Low	Medium
BoF	Crafted ICMP packets could exploit BoF in core network Control Plane interface	Remote Code Execution	Medium	Low
BoF	PEVRF buffers could be exhausted if client route tables are larger than the PE memory can handle	Remote Code Execution (CE<->PE attachment)DoS	Medium	Low
BoF	PEVRF buffers could be exhausted if client route tables are larger than the PE memory can handle	Remote Code Execution DoS	Medium	Hi
MitM	Improper label assignment on PE allows attacker to manipulate tunnel access	Privilege Escalation Privacy Integrity loss	Hi	Low
BoF	PEVRF buffers could be exhausted or malformed CE route information could be passed up to SDN controller/VNF	Remote Code Execution (CE<->PE attachment)DoS	Medium	Hi
MitM	SouthboundAPI calls can be intercepted, mangled, forged, or replayed noSSL/TLS	Privilege Escalation	Hi	Medium
RemoteCode Execution	Commodity HW/OS remote exploit on SDN Controller	PrivilegeEscalation Remote Code Execution	Hi	Low

6.1.3. Results

The results in this section demonstrate how comparison between architectures can be accomplished empirically using the security metrics presented in Section 2. Initial parameters did not produce attack graphs for the final state model. The implication is that this architecture was 'secure' given the identified vulnerabilities, attacker origin, and target.

To continue our analysis we conducted 'what-if' testing by introducing hypothetical vulnerabilities into the models to represent as yet unknown attacks against specific infrastruc-

ture services and devices. Testing automation allowed us to run and collect results from 20 competing models during this stage of the analysis.

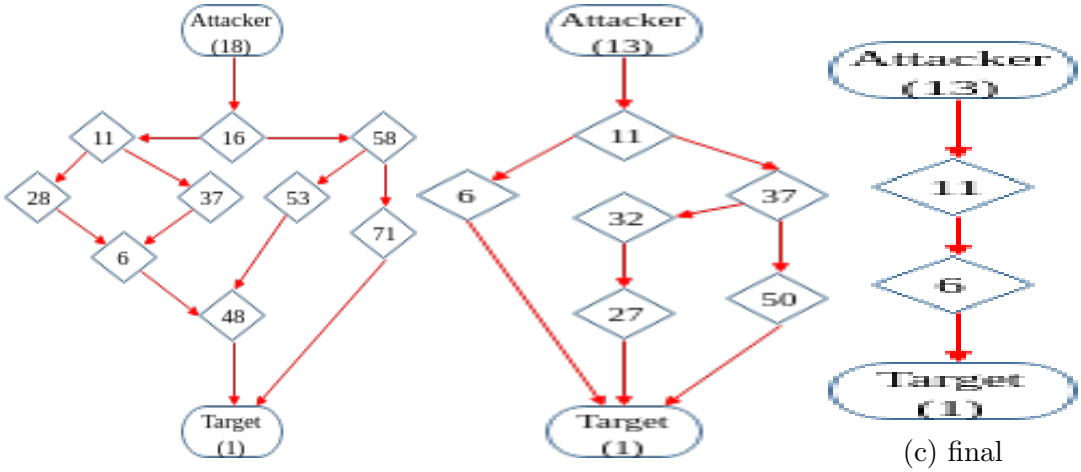
6.1.3.1. Structural Metrics

Our findings from the structural algorithms for the three network models under test can be found in Table 6.5. The shortest path metric is a measure of the path of least resistance from an attacker's origin to the target, and can be considered a priority when identifying risk in a network.

Table 6.5: Structural Metric Results Summary

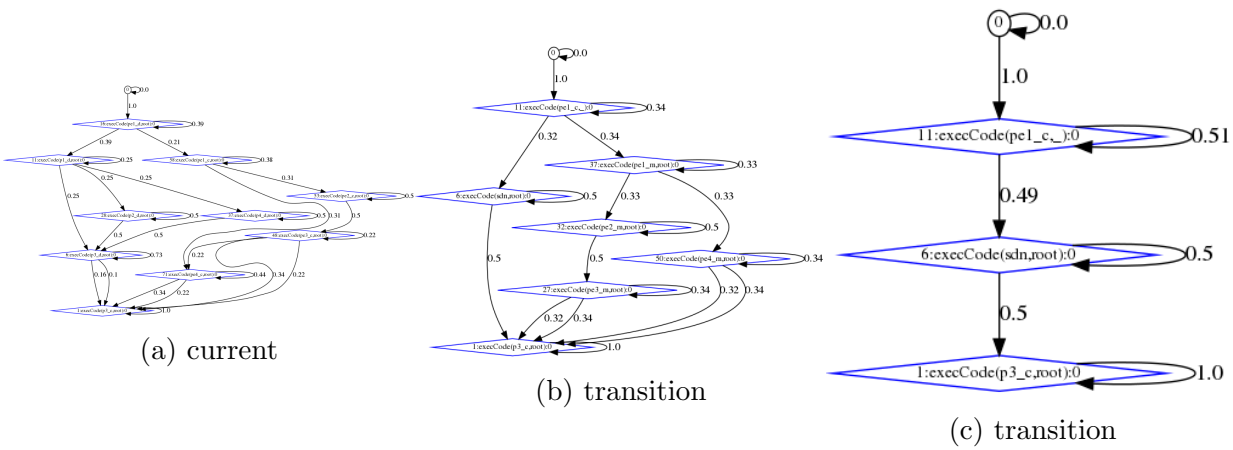
Structural Path Metric	Current	Transition	Final
Shortest Path (SP)	4	3	3
Number of Paths (NP)	6	3	1
Mean Path Length (MPL)	5.33	4	3

Node Ranking (NR):



(a) current (b) transition (c) final

Figure 6.2: Generated Attack Graphs



(a) current (b) transition (c) transition

Figure 6.3: Node Rank Analysis

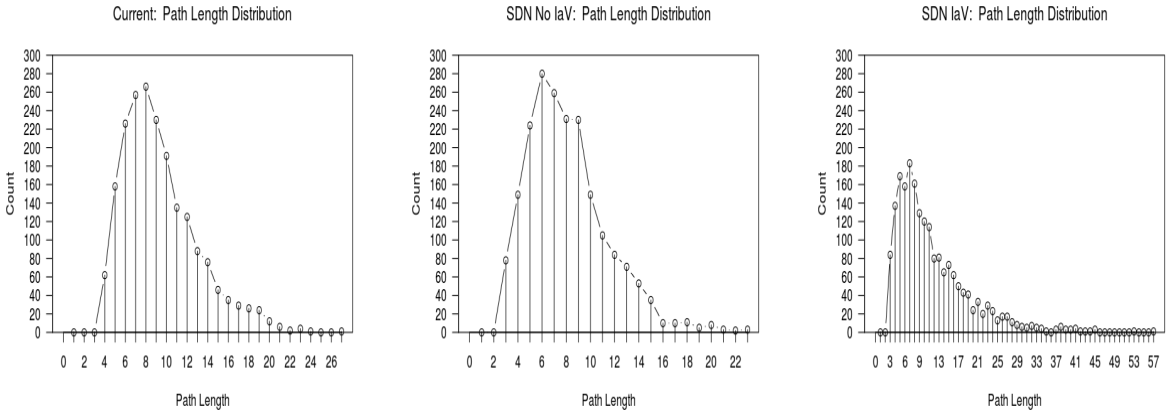
6.1.3.2. Expected Path Length

Expected Path Length (EPL):

EPL describes how long we can expect an attacker to be in our network before the target is successfully compromised. Table 6.6 shows the EPL values for each model along with the path length histogram of the simulations that were run. This histogram counts how many times the simulation reached the target in exactly Path Length steps. We can infer that the higher the EPL value, the longer an attacker will attempt to advance to the target, and the more chance we have to observe and act in response [1]. The long tail on the final state histogram indicates that, in several simulations, the observed path length nearly tripled that of the other two models. We notice that, despite having significantly more available attack paths (NP(current)=6 vs NP(SDN)=3) the expected path length of the current model is actually higher than that of the SDN model. Likewise, although the SDN models both have Shortest Path scores = 3, we have shown the resiliency of these networks to be unequal. In doing so we demonstrate the additional insight provided by incorporating vulnerability awareness into our threat modelling and planning tools.

Table 6.6: Expected Path Length Results

Current	Transition	Final
Expected Length: 9.398	Expected Length: 8.0875	Expected Length: 11.1405



6.1.3.3. Probabilistic Path

Probabilistic Path (PP):

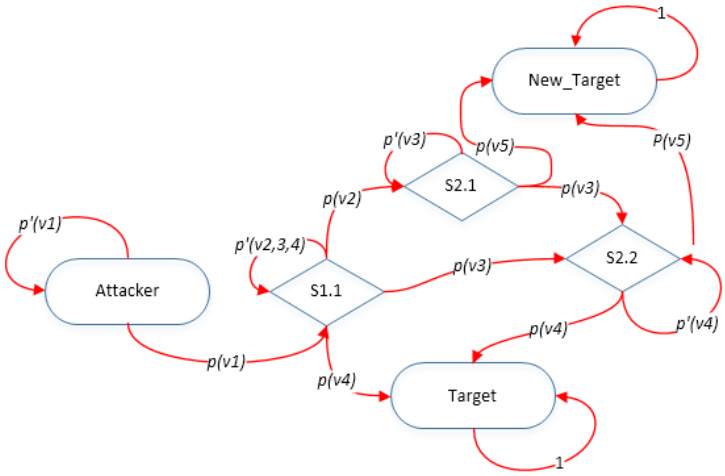


Figure 6.4: Transition diagram with additional absorbing states New_Target

In the current scenario calculating this PP metric results in a column of 1's since we only define a single 'Target' node in each of our models with 100% chance of absorption from any transient node by definition.

However it is easy to imagine a case where multiple targets are specified in the model. For example, we have added a second absorbing state, 'New_Target' to the transition diagram from Figure 2.30. This could represent a hot failover clone of the existing Target or it could be a completely separate system with unique vulnerabilities. In either case we can make a grounded prediction on which state will absorb the attacker with the highest likelihood and prepare accordingly.

6.1.4. Conclusions

In this case study we provided an end-to-end scenario demonstrating the application of the Cyber Security Analytics Framework to network migration planning. In this scenario we created distinct network models to represent point-in-time snapshots of the migration to an SDN controlled core network. The first model distills the existing architecture into a reduced set of interconnected network elements comprised of the types of services and protocols in use today. The second model modifies the first by introducing centralized SDN control and supporting infrastructure while maintaining the logical topology and control plane services of the current architecture. Specifically, we maintain large ingress ACLs on both networks to prevent unauthorized traffic from reaching core infrastructure. The final network model represents the same SDN network infrastructure while implementing an MPLS overlay to isolate publicly addressable core infrastructure nodes from internet or customer edge originating attacks.

We developed a modular, automated implementation of the CSAF that is instrumented for customization. To streamline further research in this area, the following contributions were made:

- Infrastructure setup, provisioning, analysis, and reporting are implemented using industry standard open source tools, allowing testing to run locally or on the cloud with a single command.
- Multiple end-to-end tests can be run in sequence or parallel as dictated by available resources.

- Multiple network models can be specified for a test run while remaining logically organized and version controlled.
- Multiple custom rules sets can be applied individually or grouped to a test run, allowing results to reflect the subset of rules relevant to the analysis.
- Vulnerabilities can now be weighted individually, by class, or by effect to facilitate 'what-if' analysis. In this context it is feasible to run a battery of tests against the provided architecture models in which vulnerabilities are applied stochastically and the security metrics are returned as heuristics.
- Transition matrix weighting strategies have been parameterized, allowing optimization or comparison of results.
- Metrics are extensible, customizable, and currently supported in **R** and **Python**.

During the course of this project we came across some questions that are currently being investigated.

When a network model prevents an attacker from reaching the goal, either because the model is truly secure or because no relevant vulnerability is defined in the rules or applied to the system, then no attack paths exist and subsequently no further analysis is conducted. When this occurred in our scenario above, the outcome was noted, and hypothetical vulnerabilities were introduced to applicable models to allow comparison. In a non-planning (i.e., operational) environment, there must be a mechanism to delineate the true positive null results indicating an unreachable target from the false positive null results caused by a limited rule set. We are encouraged by the work presented in [384] as a means to bound this problem to the OSI layer. However, after working with our own network infrastructure attack rules and with those published by the research community, it became clear that NVD entries don't always provide the information necessary to determine that an infrastructure attack is possible, or if it exists at all. While we can encode the conditions necessary to, for example, spoof an ARP response within a subnet, we are left without vetted CVSS exploitability and impact measures if the exploit isn't tied to a specific piece of software. In this work we provide the user a mechanism to define these scores for a general vulnerability class

or for a single instance of that vulnerability, so using the CVSS calculator with a knowledge of the system under test should yield reasonable estimates. As part of ongoing research we also allow for custom weighting strategies for cases where CVSS is not applicable.

The use of attack graphs to capture relationships between vulnerabilities, both real and hypothetical, goes beyond looking at vulnerability data in isolation to provide a powerful means for advanced analysis of the entire network with a suite of metrics catered to our mission.

Chapter 7

Future Work

7.1. Conclusions & Future Work

System security metrics are valuable only if they can produce timely, actionable measurements. In this thesis we have demonstrated a path forward for developing, testing, validating, and integrating security metrics into the full life cycle of a system.

In Chapter 2 we present the current state of security metrics. We list the working taxonomies that these metrics can be categorized by, and elaborate on the distinctions that lead to confusion when discussing security measurements. We then review modeling techniques and how these models can isolate the security properties of a system we intend to measure.

Chapter 3 presents our unified framework for security measurement and analysis, including the model for implementing individual metrics and the infrastructure built around these metrics to drive automation in a variety of scenarios. Here we establish the security metric inheritance hierarchy and enumerate properties common to all metric types and those specific to each metric subtype. We provide our extensions to attack models that expand the range of systems that can be represented. We describe how we implemented automation from the view points of a security researcher or measurement analyst, and develop our concept of security metrics as a service, *S-MaaS*, with considerations for deployment in a continuous integration or stream processing environment.

By enforcing the 4 stage pipeline abstraction, we achieve several benefits. Each phase is modular so that replacing any piece in the pipeline is straight forward. By plugging in the static reference set to the input phase we create a *unit test* of sorts for our metric library, SecMet. With PTaH, the preprocessing and transformation handlers described above, we can articulate how any or all of the security metrics will behave under a variety of conditions

for any given input - not just the reference set we describe above. This, in theory at least, should make characterizing the behaviours of security metrics on internal or sensitive systems as simple as adding input adapters to the existing set, which already includes Nessus, OVAL, NVD, CVE, and now SSFNet. By replacing our validation PTaH with whatever workflow execution engine is already in place, Apache Beam or Storm for example, the SecMet catalog becomes a drop in security measurement aid to support SecDevOps which we refer to as Security Metrics as a Service (S-MaaS).

Our takeaways from the experiments described above indicate that, while validating security metrics is not done rigorously in many of the publications, a mechanism for validation and analysis is not out of reach. We covered several scenarios of systematic security metric evaluation, but there are far more examples which we were unable to address here. By streamlining the development and evaluation process with automation, we aim to lower the barrier to entry in the field and allow researchers to spend more time developing and analyzing security metrics, which in turn should result in more secure systems being deployed.

Chapter 4 develops our solution to the lack of validation in the field of security metrics. We establish a set of validation criteria that are needed for acceptance of any metric. We define a fixed set of models that set a frame of reference for evaluating security metrics, and explain how these models can be used to isolate key properties of interest. We investigate the instrumentation needed to validate our metrics in a general manner, and implement this validation framework as extensions to an industry accepted benchmarking tool to maximize the audience and reduce friction to entry. Finally we demonstrate our enhancements built around benchmarking to automate the process of executing tests, analyzing results, and alerting on anomalies and outliers that are uncovered during large scale or long running tests.

Chapter 6 presents a case study conducted as part of AT&T's planning for infrastructure migration. The study applies the CSAF[10] pipeline to hypothetical network architectures and gives insight into how model based security metrics can be used to rank an analysis of competing alternatives. As this study occurred early in the research phase, it had a great

impact on the direction this thesis has taken. With the benefit of hindsight we are able to demonstrate both the contributions made during that initial work as well as the progress that has been made since it was completed.

Figure 7.1: General Progression and Direction of Thesis

Figure 7.1 captures with broad strokes the path our research has followed and the direction it is heading, while the timeline in Figure 7.2 summarizes previous research items that support this thesis. Listed along the top are the two long running projects that have provided both requirements and solutions in this work. SDN Migration Analytics is the focus of the case study in Chapter 6 while the Cloud Benchmarking research has provided an in depth knowledge of designing and validating cyber measurement instruments. Immediately below the long running projects are short term studies conducted over summers each year. The Tactical Edge work that bookends the summer research items focused on evaluating the security of non traditional network architectures and drove our requirement to validate metrics outside of the enterprise domains commonly found in the literature. The Maru research over the summer of 2017 and 2018 led to the development of a distributed streaming analytics system run from within a hardware trusted enclave, which forms the basis of the S-MaaS architecture described in **??**. On the right side a legend designates presentations, posters, and papers delivered that relate to the research topics listed above.

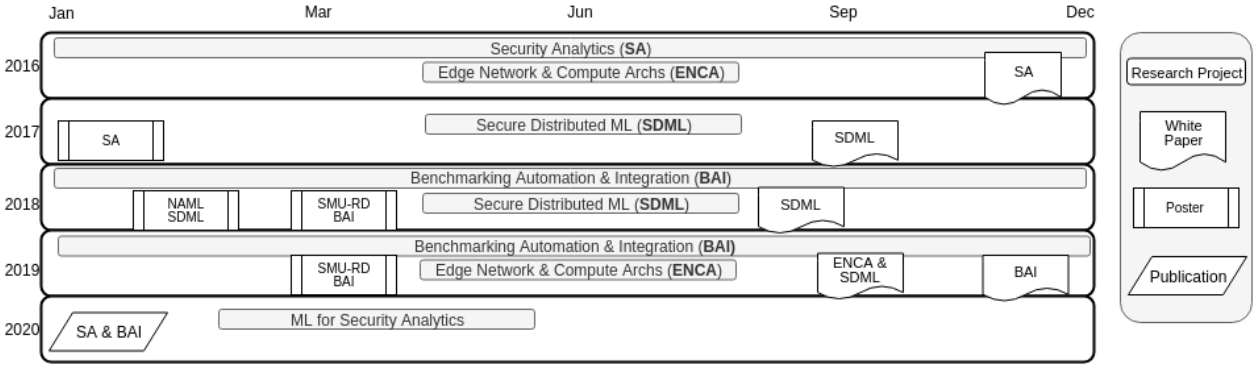


Figure 7.2: Timeline of Work Supporting Thesis

Bibliography

[1] (2005). Citation Key: unknown2005a tex.citation-number: 64. URL: http://disi.unitn.it/.

[2] *National Science and Technology Council* (2011). Citation Key: unknown2011a. URL: https://www.nitrd.gov/SUBCOMMITTEE/csia/Fed%5C_Cybersecurity%5C_RD%5C_Strategic%5C_Plan%5C_2011.pdf..

[3] (). URL: http://tnlandforms.us/cs594-cns96/attacktrees.pdf.

[4] (). URL: https://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c21.pdf.

[5] 2, 2 (), 6.

[6] "National vulnerability database (). Citation Key: unknown-a tex.citation-number: 45 tex.type: [Online]. URL: https://nvd.nist.gov/.

[7] 14:00-17:00 ISO/IEC 27004:2016. *ISO* (). URL: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/41/64120.html.

[8] Abedin, M., ET AL. Vulnerability analysis for evaluating quality of protection of security policies. In: *Proceedings of the 2Nd ACM workshop on quality of protection, ser. QoP '06*. Citation Key: abedin2006a tex.citation-number: 124. ACM, 2006, 49–52.

[9] Abraham, S. Cyber-security analytics: Stochastic models for security quantification. Citation Key: abraham2016a tex.citation-number: 78. PhD thesis. Southern Methodist University, June 2016, 467–472.

[10] Abraham, S., AND Nair, S. A novel architecture for predictive cybersecurity using non-homogenous markov models. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. IEEE, 2015, 774–781.

[11] Abraham, S., AND Nair, S. A predictive framework for cyber security analytics using attack graphs. *arXiv preprint arXiv:1502.01240* (2015).

[12] Abraham, S., AND Nair, S. Comparative analysis and patch optimization using the cyber security analytics framework. *The Journal of Defense Modeling and Simulation* 15, 2 (2018), 161–180.

[13] Abraham, S., AND Nair, S. Cyber security analytics: a stochastic model for security quantification using absorbing markov chains. *Journal of Communications* 9, 12 (2014), 899–907.

[14] Abraham, S., AND Nair, S. Exploitability analysis using predictive cybersecurity framework. In: *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)*. IEEE, 2015, 317–323.

[15] Abraham, S., AND Nair, S. Predictive cyber-security analytics framework: a non-homogenous markov model for security quantification. *arXiv preprint arXiv:1501.01901* (2015).

[16] Abubakar, A. I., ET AL. A Review of the Advances in Cyber Security Benchmark Datasets for Evaluating Data-Driven Based Intrusion Detection Systems. *Procedia Computer Science* 62 (2015), 221–227. ISSN: 18770509. DOI: 10.1016/j.procs.2015.08.443.

[17] Acosta, J. C., Padilla, E., AND Homer, J. Augmenting attack graphs to represent data link and network layer vulnerabilities. In: *MILCOM 2016 - 2016 IEEE Military Communications Conference*. Nov. 2016, 1010–1015. DOI: 10.1109/MILCOM.2016.7795462.

[18] Ahmed, A., ET AL. Distributed large-scale natural graph factorization. In: *Proceedings of the 22nd international conference on World Wide Web*. ACM. 2013, 37–48.

[19] Ahmed, M., Al-Shaer, E., AND Khan, L. A novel quantitative approach for measuring network security. In: *IEEE INFOCOM'2008.Google scholar*. Citation Key: ahmed2008a. IEEE, Apr. 2008, 1957–1965.

[20] Ahmed, M., ET AL. Objective risk evaluation for automated security management. *Journal of Network and Systems Management* 19, 3 (Sept. 2011). Citation Key: ahmed2011a tex.citation-number: 126, 343–366.

[21] Akyildiz, I., ET AL. A survey on sensor networks. *IEEE Communications Magazine* 40, 8 (Aug. 2002). Citation Key: akyildiz2002a tex.citation-number: 137, 102–114.

[22] Albanese, M., Jajodia, S., AND Noel, S. Time-efficient and cost-effective network hardening using attack graphs. In: *Proc. IEEE*. Vol. DSN'12. Citation Key: albanese2012a. 2012, 1–12.

[23] Albert, R., AND Barabasi, A. Statistical mechanics of complex networks. *Rev. Mod. Phys* 74 (2002). Citation Key: albert2002a.

[24] Alhazmi, O. H., AND Malaiya, Y. K. Application of vulnerability discovery models to major operating systems. *IEEE Transactions on Reliability* 57, 1 (2008). tex.ids: alhazmiApplicationVulnerabilityDiscovery2008a, alhazmiApplicationVulnerabilityDiscovery2008b, 14–22.

[25] Almasizadeh, J. Intrusion process modeling for security quantification. In: *2009 international conference on availability, reliability and security*. Citation Key: almasizadeh2009b tex.citation-number: 74. IEEE, 2009, 114–121.

[26] Almasizadeh, J., AND Azgomi, M. A method for estimation of the success probability of an intrusion process by considering the temporal aspects of the attacker behavior. In: *Transactions on computational science IV*. Citation Key: almasizadeh2009a tex.citation-number: 73. 2009, 200–214.

[27] Almasizadeh, J., AND Azgomi, M. Mean privacy: A metric for security of computer systems. *Computer Communications* 52 (Oct. 2013). Citation Key: almasizadeh2013b tex.citation-number: 121, 47–59.

[28] Almasizadeh, J., AND Azgomi, M. A. A stochastic model of attack process for the evaluation of security metrics. *Computer Networks*. Towards a Science of Cyber

Security 57, 10 (July 2013). tex.ids: almasizadeh2013a tex.citation-number: 40, 2159–2180. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2013.03.011.

[29] Amin, S., Schwartz, G. A., AND Hussain, A. In quest of benchmarking security risks to cyber-physical systems. *IEEE Network* 27, 1 (Jan. 2013), 19–24. ISSN: 0890-8044. DOI: 10.1109/MNET.2013.6423187.

[30] Ammann, P., Wijesekera, D., AND Kaushik, S. Scalable, graph-based network vulnerability analysis. In: *Proc. ACM*. Vol. CCS'02. Citation Key: ammann2002a tex.citation-number: 91. 2002, 217–224.

[31] Anderson, R. Why information security is hard-an economic perspective. In: *Seventeenth Annual Computer Security Applications Conference*. tex.ids: andersonWhyInformationSecurity2001a, andersonWhyInformationSecurity2001b. IEEE, 2001, 358–365.

[32] Anderson, R., AND Moore, T. Information Security Economics – and Beyond (), 24.

[33] Anderson, R., AND Moore, T. The Economics of Information Security: A Survey and Open Questions (2007), 27.

[34] Anisetti, M., ET AL. A Security Benchmark for OpenStack. In: *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, June 2017, 294–301. ISBN: 978-1-5386-1993-3. DOI: 10.1109/CLOUD.2017.45. URL: http://ieeexplore.ieee.org/document/8030601/.

[35] Association, C. R. Four grand challenges in trustworthy computing. *Tech. Rep* (2003). Citation Key: association2003a tex.citation-number: 144. URL: http://archive.cra.org/reports/trustworthy.computing.pdf.

[36] Avizienis, A., Laprie, J.-c., AND Randell, B. Fundamental concepts of dependability. In: *LAAS-CNRS N01145, Tech.* Citation Key: avizienis2001a tex.citation-number: 49. Rep, 2001.

[37] Awduche, D. O., AND Agogbua, J. Requirements for traffic engineering over MPLS (1999).

[38] Axelsson, S. The base-rate fallacy and its implications for the difficulty of intrusion detection. In: *Proc. ACM CCS'09*. Citation Key: axelsson2009a. 2009, 1–7.

[39] Bacciu, D., ET AL. A Gentle Introduction to Deep Learning for Graphs. *arXiv:1912.12693 [cs, stat]* (Dec. 2019). arXiv: 1912.12693. URL: http://arxiv.org/abs/1912.12693.

[40] Bacic, E., Froh, M., AND Henderson, G. MulVAL Extensions for Dynamic Asset Protection (Apr. 2006), 68.

[41] Backes, M., AND Nürnberger, S. Oxymoron: Making fine-grained memory randomization practical by allowing code sharing. In: *Proc. USENIX security symposium*. Citation Key: backes2014a. 2014, 433–447.

[42] Barik, M. S., Sengupta, A., AND Mazumdar, C. Attack graph generation and analysis techniques. *Defence science journal* 66, 6 (2016). tex.ids: barik2016a tex.citation-number: 94 publisher: Defence Scientific Information & Documentation Centre, 559.

[43] Basili, V., Briand, L., AND Melo, W. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22, 10 (Oct. 1996), 751–761. ISSN: 00985589. DOI: 10.1109/32.544352.

[44] Bayuk, J. L. Security as a theoretical attribute construct. *Computers & Security* 37 (Sept. 2013). tex.ids: bayuk2013a tex.citation-number: 11, 155–175. ISSN: 01674048. DOI: 10.1016/j.cose.2013.03.006.

[45] Beck, K. M., ET AL. Manifesto for Agile Software Development. In: 2013.

[46] Belkin, M., AND Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. In: *Advances in neural information processing systems*. 2002, 585–591.

[47] Bell, D., AND LaPadula, L. *Secure computer systems: Mathematical foundations (volume 1)*. 1973.

[48] Bellovin, S. On the Brittleness of Software and the Infeasibility of Security Metrics. *IEEE Security & Privacy Magazine* 4, 4 (July 2006). tex.ids: bellovinBrittlenessSoftwareInfeasibility2006a, 96–96. ISSN: 1540-7993. DOI: 10.1109/MSP.2006.101.

[49] Benenson, Z., Kühn, U., AND Lucks, S. *Cryptographic attack metrics*. Ed. by Dependability Metrics, I., Freiling, F., AND Reussner, R. Citation Key: benenson2008a tex.citation-number: 44. Springer, 2008.

[50] Berinato, S. Finally, a real return on security spending (2002). Citation Key: berinato2002a. URL: http://www.cio.com/article/2440999/metrics/finally--a-real-return-on-security-spending.html..

[51] Biggio, B., Fumera, G., AND Roli, F. Security evaluation of patternclassifiers under attack. *IEEE Trans. Knowl. Data Eng* 26, 4 (2014). Citation Key: biggio2014a, 984–996.

[52] Bilge, L., AND Dumitras, T. Before we knew it: An empirical study of zero-day attacks in the real world. In: *Proc. ACM CCS'12*. Citation Key: bilge2012a tex.citation-number: 135. 2012, 833–844.

[53] Bishop, M. What is computer security? *IEEE Security & Privacy Magazine* 1, 1 (Jan. 2003). Citation Key: bishop2003a tex.citation-number: 5, 67–69.

[54] Bo, Y., Lin, Y., AND Ru, M. Quality of protection in web service: An overview. In: *Instrumentation, measurement, computer, communi- cation and control, 2011 first international conference on*. Citation Key: bo2011a tex.citation-number: 50. Oct. 2011, 495–498.

[55] Boggs, N., Du, S., AND Stolfo, S. Measuring drive-by download defense in depth. In: *Proc. RAID'14. 172–191.Google scholar*. Citation Key: boggs2014a. 2014.

[56] Boggs, N., AND Stolfo, S. ALDR: A new metric for measuring effective layering of defenses. In: *Proc. Layered assurance workshop (LAW'11*. Citation Key: boggs2011a. 2011.

[57] Böhme, R., AND Freiling, F. On Metrics and Measurements (2008). Ed. by Dependability Metrics, I., Freiling, F., AND Reussner, R. Citation Key: boehme2008b.

[58] Böhme, R., AND Félegyházi, M. Optimal information security investment with penetration testing. In: *International Conference on Decision and Game Theory for Security*. Springer, 2010, 21–37.

[59] Bohme, R., AND Moore, T. Security Metrics and Security Investment (2013), 36.

[60] Böhme, R., AND Nowey, T. Economic Security Metrics. In: *Dependability Metrics*. Ed. by Eusgeld, I., Freiling, F. C., AND Reussner, R. Vol. 4909. Citation Key: boehme2008a. Springer Berlin Heidelberg, 2008, 176–187. ISBN: 978-3-540-68946-1. DOI: 10.1007/978-3-540-68947-8_15. URL: http://link.springer.com/10.1007/978-3-540-68947-8%5C_15.

[61] Bonneau, J. Statistical metrics for individual password strength. In: *Proc. International conference on security protocols*. Citation Key: bonneau2012a. 2012, 76–86.

[62] Bonneau, J. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In: *Proc. IEEE symposium on security and privacy*. Citation Key: bonneau2012b. 2012, 538–552.

[63] Borbor, D., ET AL. Diversifying net- work services under cost constraints for better resilience against unknown attacks. In: *30th annual IFIP WG 11.3 conference, DBSec*. Citation Key: borbor2016a tex.citation-number: 117. Springer International Publishing, 2016, 295–312.

[64] Bos, H. The cyber security body of knowledge. In: tex.chapter: Operating Systems & Virtualisation. University of Bristol, 2019. URL: https://www.cybok.org/.

[65] Boutaba, R., ET AL. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications* 9, 1 (June 2018), 16. ISSN: 1869-0238. DOI: 10.1186/s13174-018-0087-2.

[66] Boyer, W., AND McQueen, M. Ideal based cyber security technical metrics for control systems. In: *Proceedings of the 9th interna- tional conference on critical information infrastructures security (CRITIS'07*. Citation Key: boyer2007a tex.citation-number: 20. 2007.

[67] Bruna, J., ET AL. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]* (May 2014). arXiv: 1312.6203. URL: http://arxiv.org/abs/1312.6203.

[68] Buczak, A. L., AND Guven, E. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials* 18, 2 (2016), 1153–1176. ISSN: 2373-745X. DOI: 10.1109/COMST.2015.2494502.

[69] Burnap, P. The cyber security body of knowledge. In: tex.chapter: Risk Management & Governance. University of Bristol, 2019. URL: https://www.cybok.org/.

[70] Burow, N., ET AL. Control-flow integrity: Precision, security, and performance (2016). Citation Key: burow2016a.

[71] Burr, W., Dodson, D., AND Polk, W. Electronic authentication guideline (2006). Citation Key: burr2006a tex.type: NIST publication 800-63 version 1.0.2. URL: http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1%5C_0%5C_2.pdf.Google.

[72] C. Villarrubia, E. F.-M., AND Mario, P. Towards a classification of security metrics. In: *Proceedings of the 2nd international workshop on security in information systems (WOSIS 2004). In con- junction with ICEIS*. Citation Key: c2004a tex.citation-number: 18. 2004, 342–350.

[73] C.I.S. The CIS security metrics (2010). Citation Key: c2010a. URL: http://benchmarks.cisecurity.org/downloads/metrics/..

[74] Cao, S., Lu, W., AND Xu, Q. Grarep: Learning graph representations with global structural information. In: *Proceedings of CIKM*. 2015, 891–900.

[75] Čapkun, S. The cyber security body of knowledge. In: tex.chapter: Physical Layer & Telecommunications. University of Bristol, 2019. URL: https://www.cybok.org/.

[76] Cardenas, A., Baras, J., AND Seamon, K. A framework for the evaluation of intrusion detection systems. In: *Proc. IEEE 2006 symposium on security and privacy. Google scholar*. Citation Key: cardenas2006a. 2006.

[77] Cardenas, A. The cyber security body of knowledge. In: tex.chapter: Cyber-Physical Systems Security. University of Bristol, 2019. URL: https://www.cybok.org/.

[78] Carin, L., Cybenko, G., AND Hughes, J. Cybersecurity strategies: The QuERIES methodology. *IEEE Comput* 41, 8 (2008). Citation Key: carin2008a, 20–26.

[79] Carlini, N., AND Wagner, D. ROP is still dangerous: Breaking modern defenses. In: *Proc. USENIX security symposium*. Citation Key: carlini2014a. 2014, 385–399.

[80] Carlini, N., ET AL. Control-flow bending: On the effectiveness of control-flow integrity. In: *24th USENIX security symposium*. Citation Key: carlini2015a. 2015, 161–176.

[81] Carnavalet, X. C. D., AND Mannan, M. A large-scale evaluation of high-impact password strength meters. *ACM TISSEC* 18, 1, 1 (2015). Citation Key: carnavalet2015a.

[82] Carolina, R. The cyber security body of knowledge. In: tex.chapter: Law & Regulation. University of Bristol, 2019. URL: https://www.cybok.org/.

[83] Castelluccia, C., Dürmuth, M., AND Perito, D. Adaptive password-strength meters from markov models. In: *Proc. NDSS'12.Google scholar*. Citation Key: castelluccia2012a. 2012.

[84] Chakrabarti, A., AND Govindarasu, M. Internet infrastructure security: A taxonomy. *Network, IEEE* 16 (Dec. 2002), 13–21. DOI: 10.1109/MNET.2002.1081761.

[85] Chandola, V., Banerjee, A., AND Kumar, V. Anomaly detection: A survey. *ACM Comput. Surv* 41, 3, 1 (2009). Citation Key: chandola2009a tex.address: Google Scholar.

[86] Chandra, A. K., AND Harel, D. Horn clause queries and generalizations. *The Journal of Logic Programming* 2, 1 (1985), 1–15.

[87] Chandramouli, R., ET AL. Comprehensive Security Assurance Measures for Virtualized Server Environments. *Comprehensive Security Assurance Measures for Virtualized Server Environments* (2018), 55–77. DOI: https://doi.org/10.1007/978-3-030-04834-1_3.

[88] Chatzipoulidis, A., Michalopoulos, D., AND Mavridis, I. Information infrastructure risk prediction through platform vulnerability analysis. *Journal of Systems and Software* 106, C (Aug. 2015). Citation Key: chatzipoulidis2015a tex.citation-number: 120, 28–41.

[89] Chawla, N. V., ET AL. SMOTEBoost: Improving prediction of the minority class in boosting. In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2003, 107–119. URL: http://link.springer.com/chapter/10.1007/978-3-540-39804-2%5C_12.

[90] Chen, M. Directed Graph Embedding (), 6.

[91] Chen, Z., AND Ji, C. Measuring network-aware worm spreading ability. In: *Proc. INFOCOM'2007*. Citation Key: chen2007a. 2007, 116–124.

[92] Cheng, P., ET AL. Aggregating CVSS Base Scores for Semantics-Rich Network Security Metrics. In: *2012 IEEE 31st Symposium on Reliable Distributed Systems*. tex.ids: cheng2012a. IEEE, Oct. 2012, 31–40. ISBN: 978-1-4673-2397-0. DOI: 10.1109/SRDS.2012.4. URL: http://ieeexplore.ieee.org/document/6424837/.

[93] Cheng, Y., ET AL. Metrics of Security. In: *Cyber Defense and Situational Awareness*. Ed. by Kott, A., Wang, C., AND Erbacher, R. F. Vol. 62. tex.ids: cheng2014a. 2014, 263–295. DOI: 10.1007/978-3-319-11391-3_13.

[94] Cherdantseva, Y. A review of cyber security risk assessment methods for SCADA systems. *Comput. Security* 56 (Feb. 2016). tex.ids: cherdantseva2016a tex.citation-number: 143., 1–27.

[95] Chew, E., ET AL. Performance Measurement Guide for Information Security (July 2008). Citation Key: chew2008a. URL: https://www.nist.gov/publications/performance-measurement-guide-information-security.

[96] Cho, J., Cam, H., AND Oltramari, A. Effect of personality traits on trust and risk to phishing vulnerability: Modeling and analysis. In: *Proc. IEEE CogSIMA'16.Google scholar*. Citation Key: cho2016a. 2016.

[97] Cho, J.-H., Hurley, P. M., AND Xu, S. Metrics and measurement of trustworthy systems. In: *MILCOM 2016 - 2016 IEEE Military Communications Conference*. IEEE, Nov. 2016, 1237–1242. ISBN: 978-1-5090-3781-0. DOI: 10.1109/MILCOM.2016.7795500. URL: http://ieeexplore.ieee.org/document/7795500/.

[98] Chowdhary, A., Pisharody, S., AND Huang, D. SDN based Scalable MTD solution in Cloud Network. In: *Proceedings of the 2016 ACM Workshop on Moving Target Defense - MTD'16*. ACM Press, 2016, 27–36. ISBN: 978-1-4503-4570-5. DOI: 10.1145/2995272.2995274. URL: http://dl.acm.org/citation.cfm?doid=2995272.2995274.

[99] CIS Controls V7 Measures & Metrics. *CIS* (). URL: https://www.cisecurity.org/white-papers/cis-controls-v7-measures-metrics/.

[100] Corporation, T. M. Structured Threat Information eXpression (STIX™) (), 18.

[101] Costa, G., Russo, E., AND Armando, A. Automating the Generation of Cyber Range Virtual Scenarios with VSDL (), 19.

[102] Council, I. Hard problem list (2007). Citation Key: council2007a. URL: http://www.infosec-research.org/.

[103] Cowie, J., Ogielski, A., AND Nicol, D. The SSFNet network simulator. *Software on-line: http://www. ssfnet. org/homePage. html* (2002).

[104] Cui, P., ET AL. A Survey on Network Embedding. *IEEE Transactions on Knowledge and Data Engineering* 31, 5 (May 2019), 833–852. ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2018.2849727.

[105] Da Silva, C. A., Ferreira, A. S., AND Geus, P. L. de A methodology for management of cloud computing using security criteria. In: *2012 IEEE Latin America Conference on Cloud Computing and Communications (LatinCloud)*. IEEE, 2012, 49–54.

[106] Da, G., Xu, M., AND Xu, S. A new approach to modeling and analyzing security of networked systems. In: *Proc. HotSoS'14. Google scholar*. Citation Key: da2014a. 2014.

[107] Dacier, M., Deswarte, Y., AND Kaâniche, M. Information Systems Secu- rity: Facing the information society of the 21st century. In: *ch. Models and tools for quantitative assessment of operational security*. Citation Key: dacier1996a tex.citation-number: 70. Springer US, 1996, 177–186.

[108] Dacier, M., AND Deswarte, Y. Privilege Graph: An Extension to the Typed Access Matrix Model. In: *Proceedings of the Third European Symposium on Research in Computer Security*. ESORICS '94. Springer-Verlag, London, UK, UK, 1994, 319–334. ISBN: 3-540-58618-0. URL: http://dl.acm.org/citation.cfm?id=646645.699167.

[109] Dacier, M., Deswarte, Y., AND Kaâniche, M. Quantitative Assessment of Operational Security: Models and Tools. *Information Systems Security, ed. by SK Katsikas and D. Gritzalis, London, Chapman & Hall* (1996). Citation Key: dacier1996b tex.citation-number: 71, 23.

[110] Dagon, D., Zou, C., AND Lee, W. Modeling botnet propagation using time zones. In: *Proc. NDSS'06.Google Scholar*. Citation Key: dagon2006a. 2006.

[111] Dagon, D., ET AL. A taxonomy of botnet structures. In: *Proc. ACSAC'07. 325–339.Google Scholar*. Citation Key: dagon2007a. 2007.

[112] Dai, H., ET AL. Adversarial Attack on Graph Structured Data. *arXiv:1806.02371 [cs, stat]* (June 2018). arXiv: 1806.02371. URL: http://arxiv.org/abs/1806.02371.

[113] Dalvi, N., ET AL. Adversarial classification. In: *Proc.* Vol. KDD'04. Citation Key: dalvi2004a. 2004, 99–108.

[114] Dantu, R., AND Kolan, P. *Risk Management Using Behavior Based Bayesian Networks*. Citation Key: dantu2005a tex.citation-number: 110. Springer, 2005.

[115] Dantu, R., Kolan, P., AND Cangussu, J. Network risk management using attacker profiling. *Security and Communication Networks* 2, 1 (Jan. 2009). Citation Key: dantu2009a tex.citation-number: 109, 83–96.

[116] Dantu, R., Loper, K., AND Kolan, P. Risk management using behavior based attack graphs. In: *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.* Vol. 1. tex.ids: dantu2004a tex.citation-number: 111 ISSN: null. Apr. 2004, 445–449 Vol.1. DOI: 10.1109/ITCC.2004.1286496.

[117] Davi, L., ET AL. Stitching the gadgets: On the ineffectiveness of coarse-grained control-flow integrity protection. In: *Proc. USENIX security symposium*. Citation Key: davi2014a. 2014, 401–416.

[118] De Bièvre, P. The 2007 International Vocabulary of Metrology (VIM), JCGM 200:2008 [ISO/IEC Guide 99]: Meeting the need for intercontinentally understood concepts and their associated intercontinentally agreed terms. *Clinical Biochemistry*. Highlight Section: Quality & Accreditation in Laboratory Medicine 42, 4 (Mar. 2009), 246–248. ISSN: 0009-9120. DOI: 10.1016/j.clinbiochem.2008.09.007.

[119] Debar, H. The cyber security body of knowledge. In: tex.chapter: Security Operations & Inicident Management. University of Bristol, 2019. URL: https://www.cybok.org/.

[120] DellAmico, M., Michiardi, P., AND Roudier, Y. Password strength: An empirical analysis. In: *Proc. INFOCOM'10*. Citation Key: dellamico2010a. 2010, 983–991.

[121] Denker, J. Large Automatic Learning, Rule Extraction, and Generalization (), 46.

[122] Desmedt, Y., AND Frankel, Y. Threshold cryptosystems. In: *Proc. Crypto*. Citation Key: desmedt1989a. 1989, 307–315.

[123] Dhillon, D. Developer-Driven Threat Modeling: Lessons Learned in the Trenches. *IEEE Security & Privacy* 9, 4 (July 2011), 41–47. ISSN: 1540-7993. DOI: 10.1109/MSP.2011.47.

[124] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1 (1959), 269–271.

[125] Donovan, J., AND Prabhu, K. *Building the Network of the Future: Getting Smarter, Faster, and More Flexible with a Software Centric Approach*. CRC Press, 2017.

[126] Dua, S., AND Du, X. Data Mining and Machine Learning in Cybersecurity. In: 2011. DOI: 10.1201/b10867.

[127] Duan, Q., ET AL. Private and anony- mous data storage and distribution in cloud. In: *Services computing (SCC), 2013 IEEE international conference on*. Citation Key: duan2013a tex.citation-number: 37. June 2013, 264–271.

[128] Duggan, D. P., AND Michalski, J. T. Threat Analysis Framework (), 31.

[129] Dumitras, T., AND Shou, D. Toward a standard benchmark for computer security research: the worldwide intelligence network environment (WINE). In: *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security - BADGERS '11*. ACM Press, 2011, 89–96. ISBN:

978-1-4503-0768-0. DOI: 10.1145/1978672.1978683. URL: http://portal.acm.org/citation.cfm?doid=1978672.1978683.

[130] Durumeric, Z., ET AL. The matter of heartbleed. In: *Proc. ACM IMC'14*. Citation Key: durumeric2014a. 2014, 475–488.

[131] Edwards, B., Hofmeyr, S., AND Forrest, S. Hype and heavy tails: A closer look at data breaches. In: *Proc WEIS'15. 67–78.Google Scholar*. Citation Key: edwards2015a. 2015.

[132] Ellison, C. Ceremony Design and Analysis (), 17.

[133] Eskridge, T., ET AL. VINE: A cyber emulation environment for MTD experimentation. In: *Proc. ACM MTD'15*. Citation Key: eskridge2015a. 2015, 43–47.

[134] Evans, D. NSF/IARPA/NSA Workshop on the Science of Security. *Also see http://sos. cs. virginia. edu/., University of Virginia* (2008).

[135] F.I.R.S.T. Forum of incident response and security teams: Common vulnerability scoring system (CVSS) version 3.0 (2015). Citation Key: f2015a. URL: https://www.first.org/cvss..

[136] Fahl, S. The cyber security body of knowledge. In: tex.chapter: Web & Mobile Security. University of Bristol, 2019. URL: https://www.cybok.org/.

[137] Fleming, P. J., AND Wallace, J. J. How not to lie with statistics: the correct way to summarize benchmark results. *Communications of the ACM* 29, 3 (Mar. 1986), 218–221. ISSN: 00010782. DOI: 10.1145/5666.5673.

[138] Foley, S., ET AL. Multilevel security and quality of protection. In: *Quality of protection workshop at ESORICS 2005, the flagship european symposium on research in computer security*. Citation Key: foley2005a tex.citation-number: 56. Springer US, 2005, 93–105.

[139] Ford, M., ET AL. Implementing the advise security modeling formalism in mobius. In: *2013 43rd annual IEEE/IFIP international conference on dependable systems and networks (DSN*. Citation Key: ford2013a tex.citation-number: 128. June 2013, 1–8.

[140] Franco Rosa, F. de, AND Jino, M. A Survey of Security Assessment Ontologies. In: *Recent Advances in Information Systems and Technologies*. Ed. by Rocha, Á., ET AL. Vol. 569. Springer International Publishing, 2017, 166–173. ISBN: 978-3-319-56534-7. DOI: 10.1007/978-3-319-56535-4_17. URL: http://link.springer.com/10.1007/978-3-319-56535-4%5C_17.

[141] Franks, J., ET AL. HTTP authentication: Basic and digest access authentication (1999). Citation Key: franks1999a tex.citation-number: 60 tex.type: Internet RFC 2617,

[142] Frei, S., AND Feb, T. The security exposure of SOoftware portfolios (2010). Citation Key: frei2010a. URL: https://secunia.com/gfx/pdf/Secunia%5C_RSA%5C_Software%5C_Portfolio%5C_Security%5C_Exposure.pdf..

[143] Frigault, M., ET AL. Measuring network security using dynamic bayesian network. In: *Proc. QoP'08*. Citation Key: frigault2008a tex.ids: frigault2008b tex.citation-number: 106. 2008, 23–30.

[144] Frigault, M., AND Wang, L. Measuring Network Security Using Bayesian Network-Based Attack Graphs. In: *Computer software and applications, 2008. COMPSAC '08. 32nd annual IEEE international*. tex.ids: frigault2008b, frigaultMeasuringNetworkSecurity2008a. Jan. 2008, 698–703. ISBN: 978-0-7695-3262-2. DOI: 10.1109/COMPSAC.2008.88.

[145] Al-Fuqaha, A., ET AL. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials* 17, 4 (2015). Citation Key: al-fuqaha2015a tex.citation-number: 138, 2347–2376.

[146] Gaffney Jr, J., AND Ulvila, J. Evaluation of intrusion detectors: A decision theory approach. In: *Proc. IEEE symposium on security and privacy*. Citation Key: gaffney2001a. 2001, 50–61.

[147] Gerstel, O., AND Sasaki, G. Quality of protection (QoP): a quanti- tative unifying paradigm to protection service grades. In: *Proc. SPIE 4599, OptiComm 2001: Optical networking and communications*. Citation Key: gerstel2001a tex.citation-number: 63. Aug. 2001, 12.

[148] Ghosh, S., AND Bhattacharya, P. Analytical framework for measuring network security using exploit dependency graph. *IET Information Security* 6, 4 (Dec. 2012). Citation Key: ghosh2012a tex.citation-number: 112, 264–270.

[149] Göktas, E., ET AL. Out of control: Overcoming control-flow integrity. In: *Proc. IEEE Security and Privacy*. Citation Key: goektas2014a. 2014, 575–589.

[150] Gollmann, D. The cyber security body of knowledge. In: tex.chapter: Authentication, Authorisation & Accountability. University of Bristol, 2019. URL: https://www.cybok.org/.

[151] Gordon, L., AND Loeb, M. Budgeting process for information security expenditures. *Communications of the ACM* 49, 1 (2006). Citation Key: gordon2006a, 121–125.

[152] Gordon, L. A., AND Loeb, M. P. The economics of information security investment. *ACM Transactions on Information and System Security* 5, 4 (), 20.

[153] Goyal, P., AND Ferrara, E. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems* 151 (July 2018). arXiv: 1705.02801, 78–94. ISSN: 09507051. DOI: 10.1016/j.knosys.2018.03.022.

[154] Granjal, J., Monteiro, E., AND Silva, J. Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Communications Surveys Tutorials* 17, 3 (2015). Citation Key: granjal2015a tex.citation-number: 139, 1294–1312.

[155] Grover, A., AND Leskovec, J. node2vec: Scalable feature learning for networks. In: *Proceedings of KDD*. 2016, 855–864.

[156] Grubesic, T. H., ET AL. Comparative approaches for assessing network vulnerability. *International Regional Science Review* 31, 1 (Jan. 2008), 88–112. DOI: 10.1177/0160017607308679.

[157] Gu, G., Cárdenas, A., AND Lee, W. Principled reasoning and practical applications of alert fusion in intrusion detection systems. In: *Proc. ACM ASIACCS'08*. Citation Key: gu2008a. 2008, 136–147.

[158] Gu, G., ET AL. Measuring intrusion detection capability: An information-theoretic approach. In: *Proc. AsiaCCS'06*. Citation Key: gu2006a tex.citation-number: 39. 2006, 90–101.

[159] Hahsler, M., AND Chelluboina, S. Visualizing Association Rules: Introduction to the R-extension Package arulesViz (), 24.

[160] Hall, D. *Ansible Configuration Management*. Packt Publishing, 2013. ISBN: 9781783280810.

[161] Hamilton, W. L., Ying, R., AND Leskovec, J. Representation Learning on Graphs: Methods and Applications (), 23.

[162] Han, Y., Lu, W., AND Xu, S. Characterizing the power of moving target defense via cyber epidemic dynamics. *Proc. HotSoS'14* 10, 1 (2014). Citation Key: han2014a.

[163] Hansman, S., AND Hunt, R. A taxonomy of network and computer attacks. *Computers & Security* 24, 1 (2005). tex.ids: hansman2005a tex.citation-number: 1, 31–43.

[164] Haque, S., Keffeler, M., AND Atkison, T. An Evolutionary Approach of Attack Graphs and Attack Trees: A Survey of Attack Modeling. In: *Proceedings of the International Conference on Security and Management (SAM); Athens*. The Steering Committee of The World Congress in Computer Science, Computer

Engineering AND Applied Computing (WorldComp), 2017, 224–229. URL: https://search.proquest.com/docview/2139471619/abstract/7169E9EF244F4076PQ/1.

[165] Hardy, S., ET AL. Targeted threat index: Characterizing and quantifying politically-motivated targeted malware. In: *Proc. USENIX security symposium. Google scholar*. Citation Key: hardy2014a. 2014.

[166] Hecker, A. On System Security Metrics and the Definition Approaches. In: *2008 Second International Conference on Emerging Security Information, Systems and Technologies*. tex.ids: hecker2008a tex.citation-number: 13 ISSN: 2162-2116. Aug. 2008, 412–419. DOI: 10.1109/SECURWARE.2008.37.

[167] Heinzle, B., AND Furnell, S. Assessing the feasibility of security metrics. In: *International Conference on Trust, Privacy and Security in Digital Business*. tex.ids: heinzle2013a tex.citation-number: 19. Springer, 2013, 149–160.

[168] Henderson, G., Bacic, E., AND Froh, M. Dynamic Asset Protection and Risk Management Abstraction Study (2005), 50.

[169] Herlands, W., Hobson, T., AND Donovan, P. Effective entropy: Security-centric metric for memory randomization techniques. In: *Workshop on Cyber Security Experimentation and Test. Google Scholar*. Citation Key: herlands2014a. 2014.

[170] Herrmann, D. *Complete guide to security and privacy metrics: Measuring regulatory compliance, operational resilience, and ROI*. Citation Key: herrmann2007a tex.citation-number: 38. Auerbach Publication, 2007.

[171] Hlyne, C. N. N., Zavarsky, P., AND Butakov, S. SCAP benchmark for Cisco router security configuration compliance. In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. IEEE, Dec. 2015, 270–276. ISBN: 978-1-908320-52-0. DOI: 10.1109/ICITST.2015.7412104. URL: http://ieeexplore.ieee.org/document/7412104/.

[172] Holm, H. A large-scale study of the time required to compromise a computer system. *IEEE TDSC* 11, 1 (2014). Citation Key: holm2014a, 2–15.

[173] Holm, H., Ekstedt, M., AND Andersson, D. Empirical analysis of system-level vulnerability metrics through actual attacks. *IEEE Transactions on Dependable and Secure Computing* 9, 6 (Nov. 2012). Citation Key: holm2012a tex.citation-number: 134, 825–837.

[174] Homer, J., ET AL. Aggregating vulnerability metrics in enterprise networks using attack graphs. *Journal of Computer Security* 21, 4 (Sept. 2013). tex.ids: homer2013a, homerAggregatingVulnerabilityMetrics2013a, homerAggregatingVulnerabilityMetrics2013b tex.citation-number: 102., 561–597. ISSN: 18758924, 0926227X. DOI: 10.3233/JCS-130475.

[175] Hong, J. B., Kim, D. S., AND Takaoka, T. Scalable Attack Representation Model Using Logic Reduction Techniques. In: *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. July 2013, 404–411. DOI: 10.1109/TrustCom.2013.51.

[176] Hoskins, J. G., ET AL. Learning Networks from Random Walk-Based Node Similarities. *arXiv:1801.07386 [cs]* (Jan. 2018). arXiv: 1801.07386. URL: http://arxiv.org/abs/1801.07386.

[177] Howe, A., ET AL. The psychology of security for the home computer user. In: *IEEE symp. on security and privacy*. Citation Key: howe2012a. 2012, 209–223.

[178] Huang, L., ET AL. Adversarial machine learning (). tex.ids: huang2011a, 15.

[179] Hutchins, E. M., Cloppert, M. J., AND Amin, R. M. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains (), 14.

[180] I.S.O./I.E.C. Information technology - systems security engineering - capability maturity model (SSE-CMM), ISO/IEC 21827 (2002). Citation Key: i2002a tex.address: Geneva, Switzerland tex.citation-number: 30, 132.

[181] Idika, N., Marshall, B., AND Bhargava, B. Maximizing network security given a limited budget. In: *The fifth richard tapia celebration of diversity in computing conference: Intellect, initiatives, insight, and innovations, ser. TAPIA '09*. Citation Key: idika2009a tex.citation-number: 136. ACM, 2009, 12–17.

[182] Idika, N., AND Bhargava, B. Extending attack graph-based security metrics and aggregating their application. *IEEE Transactions on dependable and secure computing* 9, 1 (2012). tex.ids: idika2012a, idikaExtendingAttackGraphBased2012 tex.citation-number: 10, 75–85.

[183] Isaksson, C., Dunham, M. H., AND Hahsler, M. SOStream: Self Organizing Density-Based Clustering over Data Stream. In: *Machine Learning and Data Mining in Pattern Recognition*. Ed. by Perner, P. Vol. 7376. Springer Berlin Heidelberg, 2012, 264–278. ISBN: 978-3-642-31536-7. URL: http://link.springer.com/10.1007/978-3-642-31537-4%5C_21.

[184] Ivanov, S., Sviridov, S., AND Burnaev, E. Understanding Isomorphism Bias in Graph Data Sets. *arXiv:1910.12091 [cs, stat]* (Oct. 2019). arXiv: 1910.12091. URL: http://arxiv.org/abs/1910.12091.

[185] Jaatun, M. Hunting for aardvarks: Can software security be measured?" in IFIP WG 8.4, 8.9/TC 5. In: *International cross-domain conference and workshop on availability, reliability, and security, CD-ARES 2012*. Citation Key: jaatun2012a tex.citation-number: 24. 2012.

[186] Jajodia, S. Topological analysis of network attack vulnerability. In: *Proceedings of the 2Nd ACM symposium on information, computer and communications security, ser. ASIACCS '07*. Citation Key: jajodia2007a tex.citation-number: 9. ACM, 2007, 2–2.

[187] Jansen, W. Directions in Security Metrics Research (Apr. 2009). tex.ids: jansen2009a, jansenDirectionsSecurityMetrics2009a. DOI:

https://doi.org/10.6028/NIST.IR.7564. URL: https://csrc.nist.gov/publications/detail/nistir/7564/final.

[188] Jaquith, A. *Security Metrics: Replacing Fear, Uncertainty, and Doubt*. tex.ids: jaquith2007a, jaquithSecurityMetricsReplacing2007a tex.citation-number: 7 googlebooksid: Af8F00gTRN4C tex.publisher: Addison-Wesley Professional. Pearson Education, Mar. 2007. ISBN: 978-0-13-271577-5.

[189] Jha, S., Sheyner, O., AND Wing, J. Two formal analys s of attack graphs. In: *Proc. IEEE CSF*. Citation Key: jha2002a tex.citation-number: 79. 2002, 49–59.

[190] Jha, S. The cyber security body of knowledge. In: tex.chapter: Network Security. University of Bristol, 2019. URL: https://www.cybok.org/.

[191] Jha, S., Sheyner, O., AND Wing, J. M. Minimization and Reliability Analyses of Attack Graphs (), 31.

[192] Jibao, L., Huiqiang, W., AND Liang, Z. Study of network security situa- tion awareness model based on simple additive weight and grey theory. In: *2006 international conference on computational intelligence and security*. Vol. 2. Citation Key: jibao2006a tex.citation-number: 51. Nov. 2006, 1545–1548.

[193] Johnson, B., ET AL. Metrics for measuring ISP badness: The case of spam. In: *Proc. FC'12. 89–97.Google scholar*. Citation Key: johnson2012a. 2012.

[194] Johnson, B., ET AL. Why don't software developers use static analysis tools to find bugs? In: *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, May 2013, 672–681. ISBN: 978-1-4673-3076-3. DOI: 10.1109/ICSE.2013.6606613. URL: http://ieeexplore.ieee.org/document/6606613/.

[195] Jonsson, E., AND Olovsson, T. A quantitative model of the security intrusion process based on attacker behavior. *IEEE Transactions on Software Engineering; New York* 23, 4 (Apr. 1997). tex.ids: jonsson1997a, jonssonQuantitativeModelSecurity1997a tex.citation-number: 69, 235–245. ISSN: 00985589. DOI: http://dx.doi.org/10.1109/32.588541.

[196] Kaaniche, M., ET AL. Empirical analysis and statistical modeling of attack processes based on honeypots. In: *IEEE/IFIP international conference on dependable systems and networks (DSN-2006*. Citation Key: kaaniche2006a tex.citation-number: 129. 2006, 119–124.

[197] Kang, Y., ET AL. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, 615–629.

[198] Kanoun, W., ET AL. Success Likelihood of Ongoing Attacks for Intrusion Detection and Response Systems. In: *2009 International Conference on Computational Science and Engineering*. Vol. 3. Aug. 2009, 83–91. DOI: 10.1109/CSE.2009.233.

[199] Kanoun, W., ET AL. Towards Dynamic Risk Management: Success Likelihood of Ongoing Attacks. *Bell Labs Technical Journal* 17, 3 (2012). tex.ids: kanoun2012a tex.citation-number: 80, 61–78. ISSN: 1538-7305. DOI: 10.1002/bltj.21558.

[200] Karjoth, G., ET AL. Service- oriented assurance comprehensive security by explicit assurances. In: *Quality of protection*. Ed. by Gollmann, D., Massacci, F., AND Yautsiukhin, A. Citation Key: karjoth2006a tex.citation-number: 62. Springer US, 2006, 13–24.

[201] Kelley, P. G., ET AL. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In: tex.ids: kelley2012a, kelleyGuessAgainAgain2012, 15.

[202] Khosravi, H., AND Anderson, T. *Requirements for separation of IP control and forwarding*. RFC 3654, November, 2003.

[203] King, G., AND Zeng, L. Logistic regression in rare events data. *Political analysis* 9, 2 (2001), 137–163.

[204] Kingma, D., AND Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014). URL: https://arxiv.org/abs/1412.6980.

[205] Kipf, T. N., AND Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv preprint arXiv:1609.02907* (2016).

[206] Kirkham, A. *Issues with Private IP Addressing in the Internet*. 2012.

[207] Knowles, W., ET AL. A survey of cyber security management in industrial control systems. *International Journal of Critical Infrastructure Protection* 9, C (June 2015). Citation Key: knowles2015a tex.citation-number: 142, 52–80.

[208] Konte, M., Perdisci, R., AND Feamster, N. ASwatch: An AS Reputation System to Expose Bulletproof Hosting ASes. In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication - SIGCOMM '15*. tex.ids: konte2015a. ACM Press, 2015, 625–638. ISBN: 978-1-4503-3542-3. DOI: 10.1145/2785956.2787494. URL: http://dl.acm.org/citation.cfm?doid=2785956.2787494.

[209] Kordy, B., Piètre-Cambacédès, L., AND Schweitzer, P. DAG-Based Attack and Defense Modeling: Don't Miss the Forest for the Attack Trees. *arXiv:1303.7397 [cs]* (Mar. 2013). arXiv: 1303.7397. URL: http://arxiv.org/abs/1303.7397.

[210] Kotenko, I., AND Doynikova, E. Security Assessment of Computer Networks Based on Attack Graphs and Security Events. In: *Information and Communication Technology*. Ed. by Linawati, ET AL. Vol. 8407. Springer Berlin Heidelberg, 2014, 462–471. ISBN: 978-3-642-55031-7. DOI: 10.1007/978-3-642-55032-4_47. URL: http://link.springer.com/10.1007/978-3-642-55032-4%5C_47.

[211] Kotenko, I., ET AL. The Ontology of Metrics for Security Evaluation and Decision Support in SIEM Systems. In: Sept. 2013, 638–645. DOI: 10.1109/ARES.2013.84.

[212] Kott, A. Towards fundamental science of cyber security. In: *Network science and cybersecurity*. Springer, 2014, 1–13.

[213] Ksiezopolski, B. QoP-ML: Quality of protection modelling language for cryptographic protocols. *Computers & Security* 31, 4 (June 2012). Citation Key: ksiezopolski2012a tex.citation-number: 58, 569–596.

[214] Kührer, M., Rossow, C., AND Holz, T. Paint It Black: Evaluating the Effectiveness of Malware Blacklists. In: *Research in Attacks, Intrusions and Defenses*. Ed. by Stavrou, A., Bos, H., AND Portokalidis, G. Vol. 8688. tex.ids: kuehrer2014a. Springer International Publishing, 2014, 1–21. ISBN: 978-3-319-11378-4. DOI: 10.1007/978-3-319-11379-1_1. URL: http://link.springer.com/10.1007/978-3-319-11379-1%5C_1.

[215] Kundu, A., ET AL. Analysis of attack graph-based metrics for quantification of network security. In: *2012 Annual IEEE India Conference (INDICON)*. Dec. 2012, 530–535. DOI: 10.1109/INDCON.2012.6420675.

[216] Kutuzov, A., ET AL. Making Fast Graphbased Algorithms with Graph Metric Embeddings. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019, 3349–3355. DOI: 10.18653/v1/P19-1325. URL: https://www.aclweb.org/anthology/P19-1325.

[217] Al-Kuwaiti, M., Kyriakopoulos, N., AND Hussein, S. A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications Surveys Tutorials* 11, 2 (2009). Citation Key: al-kuwaiti2009a tex.citation-number: 55, 106–124.

[218] Lalonde Levesque, F., ET AL. A clinical study of risk factors related to malware infections. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. tex.ids: levesque2013a. ACM Press, 2013, 97–108. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516747. URL: http://dl.acm.org/citation.cfm?doid=2508859.2516747.

[219] Lampson, B. Practical Principles for Computer Security (). tex.ids: lampson2006a, 47.

[220] Lan, F., Chunlei, W., AND Guoqing, M. A framework for network secu- rity situation awareness based on knowledge discovery. In: *2010 2nd international*

conference on computer engineering and technology. Vol. 1. Citation Key: lan2010a tex.citation-number: 67. Apr. 2010, 1–226–1–231.

[221] Landwehr, C. E., ET AL. A taxonomy of computer program security flaws. *ACM Computing Surveys* 26, 3 (Sept. 1994). tex.ids: landwehr1994a, 211–254. ISSN: 03600300. DOI: 10.1145/185403.185412.

[222] Lantz, B., Heller, B., AND McKeown, N. A network in a laptop: rapid prototyping for software-defined networks. In: *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks - Hotnets '10*. ACM Press, 2010, 1–6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868466. URL: http://portal.acm.org/citation.cfm?doid=1868447.1868466.

[223] Larsen, P., ET AL. SoK: Automated Software Diversity. In: *2014 IEEE Symposium on Security and Privacy*. tex.ids: larsen2014a. IEEE, May 2014, 276–291. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.25. URL: http://ieeexplore.ieee.org/document/6956570/.

[224] Latora, V., AND Marchiori, M. Vulnerability and Protection of Critical Infrastructures. *Physical Review E* 71, 1 (Jan. 2005). arXiv: cond-mat/0407491, 015103. ISSN: 1539-3755, 1550-2376. DOI: 10.1103/PhysRevE.71.015103.

[225] LeCun, Y., Bengio, Y., AND Hinton, G. Deep learning. *Nature* 521, 7553 (May 2015), 436–444. ISSN: 0028-0836. DOI: 10.1038/nature14539.

[226] Lee, W. The cyber security body of knowledge. In: tex.chapter: Malware & Attack Technology. University of Bristol, 2019. URL: https://www.cybok.org/.

[227] Lee, W., ET AL. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security* 10, 1/2 (Mar. 2002). tex.ids: lee2002a, 5. ISSN: 0926227X. DOI: 10.3233/JCS-2002-101-202.

[228] LeMay, E., ET AL. Model-based Security Metrics Using ADversary VIew Security Evaluation (ADVISE). In: *2011 Eighth International Conference on Quantitative*

Evaluation of SysTems. tex.ids: lemay2011a. IEEE, Sept. 2011, 191–200. DOI: 10.1109/QEST.2011.34.

[229] Leversage, D., AND Byres, E. Estimating a system's mean time- to-Compromise. *IEEE Security & Privacy Magazine* 6, 1 (2008). Citation Key: leversage2008a tex.citation-number: 76, 52–60.

[230] Levin, D. Lessons learned in using live red teams in IA experiments. In: *Proceedings DARPA Information Survivability Conference and Exposition*. tex.ids: levin2003a. IEEE Comput. Soc, 2003, 110–119. ISBN: 978-0-7695-1897-8. DOI: 10.1109/DISCEX.2003.1194877. URL: http://ieeexplore.ieee.org/document/1194877/.

[231] Li, W., AND Vaughn, R. B. Cluster security research involving the modeling of network exploitations using exploitation graphs. In: *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*. Vol. 2. Citation Key: li2006a tex.citation-number: 95. IEEE, May 2006, 26–26.

[232] Li, X., Parker, P., AND Xu, S. A Stochastic Model for Quantitative Security Analyses of Networked Systems. *IEEE Transactions on Dependable and Secure Computing; Washington* 8, 1 (Mar. 2011). Citation Key: li2011a, 28–43. ISSN: 15455971. DOI: http://dx.doi.org/10.1109/TDSC.2008.75.

[233] Li, Y., ET AL. Gated Graph Sequence Neural Networks. *arXiv:1511.05493 [cs, stat]* (Sept. 2017). arXiv: 1511.05493. URL: http://arxiv.org/abs/1511.05493.

[234] Lindqvist, U., AND Jonsson, E. How to systematically classify computer security intrusions. In: *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*. tex.ids: lindqvist1997a tex.citation-number: 2. IEEE Comput. Soc. Press, 1997, 154–163. ISBN: 978-0-8186-7828-8. DOI: 10.1109/SECPRI.1997.601330. URL: http://ieeexplore.ieee.org/document/601330/.

[235] Lippmann, R. Validating and restoring defense in depth using attack graphs. In: *Proc. IEEE mil. Commun. Conf. (MILCOM*. tex.ids: lippmann2006a tex.citation-number: 97. Oct. 2006, 1–10.

[236] Lippmann, R. P., ET AL. *Continuous Security Metrics for Prevalent Network Threats: Introduction and First Four Metrics:* tex.ids: lippmann2012a. May 2012. DOI: 10.21236/ADA565825. URL: http://www.dtic.mil/docs/citations/ADA565825.

[237] Lippmann, R. P., AND Ingols, K. W. *An annotated review of past papers on attack graphs*. Citation Key: lippmann2005a tex.citation-number: 92. 2005.

[238] Littlewood, B., ET AL. Towards Operational Measures of Computer Security. *Journal of Computer Security* 2, 2–3 (Apr. 1993). tex.ids: littlewood1993a tex.citation-number: 68. tex.publisher: ScholarDigital Library, 211–229. ISSN: 18758924, 0926227X. DOI: 10.3233/JCS-1993-22-308.

[239] Liu, C., Singhal, A., AND Wijesekera, D. A Logic Based Network Forensics Model for Evidence Analysis (), 19.

[240] Liu, Y., ET AL. Cloudy with a Chance of Breach: Forecasting Cyber Security Incidents. In: tex.ids: liu2015a, 50.

[241] Liu, Y., AND Man, H. Network vulnerability assessment using Bayesian networks. In: *Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security 2005*. Vol. 5812. tex.ids: liu2005a, liuNetworkVulnerabilityAssessment2005a, liuNetworkVulnerabilityAssessment2005b tex.citation-number: 104. International Society for Optics AND Photonics, Mar. 2005, 61–71. DOI: 10.1117/12.604240. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/5812/0000/Network-vulnerability-assessment-using-Bayesian-networks/10.1117/12.604240.short.

[242] Lowd, D., AND Meek, C. Adversarial learning. In: *Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05*. tex.ids: lowd2005a tex.publisher: Google Scholar. ACM Press, 2005, 641. ISBN:

978-1-59593-135-1. DOI: 10.1145/1081870.1081950. URL: http://portal.acm.org/citation.cfm?doid=1081870.1081950.

[243] Lu, K., ET AL. ASLR-Guard: Stopping Address Space Leakage for Code Reuse Attacks. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. tex.ids: lu2015a. ACM Press, 2015, 280–291. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813694. URL: http://dl.acm.org/citation.cfm?doid=2810103.2813694.

[244] Lu, W., Xu, S., AND Yi, X. Optimizing active cyber defense dynamics. In: *Proc. GameSec'13*. Citation Key: lu2013a. 2013, 206–225.

[245] Lunt, T. A survey of intrusion detection techniques. *Computers & Security* 12, 4 (June 1993). Citation Key: lunt1993a tex.citation-number: 6, 405–418.

[246] M.I.T.R.E. Common weakness scoring system (CWSS version 1.0.1 (2014). Citation Key: m2014a. URL: https://cwe.mitre..

[247] Maclaurin, D., Duvenaud, D. K., AND Adams, R. P. Gradient-based Hyperparameter Optimization through Reversible Learning. In: *ICML*. 2015, 2113–2122. URL: http://www.jmlr.org/proceedings/papers/v37/maclaurin15.pdf.

[248] Madan, B. B., ET AL. A method for modeling and quantifying the security attributes of intrusion tolerant systems. *Perform. Eval* 56, 1–4 (2004). Citation Key: madan2004b tex.ids: madan2004a tex.citation-number: 132.

[249] Madan, B. B., ET AL. Modeling and quantification of security attributes of software systems. In: *Proceedings International Conference on Dependable Systems and Networks*. tex.ids: madan2002a, madanModelingQuantificationSecurity2002a, madanModelingQuantificationSecurity2002b tex.citation-number: 131. IEEE, 2002, 505–514.

[250] Mahalingam, S., Abdollah, F. M., AND Sahib, S. *Learner Centric in M-Learning: Integration of Security, Dependability and Trust.* ERIC, 2014.

[251] Manadhata, P. K., AND Wing, J. M. An attack surface metric. *IEEE Transactions on Software Engineering* 37, 3 (2010). tex.ids: manadhata2011a, manadhataAttackSurfaceMetric, manadhataAttackSurfaceMetric2011, manadhataAttackSurfaceMetric2011a tex.citation-number: 32, 371–386.

[252] Marconato, G. V., Kaâniche, M., AND Nicomette, V. A vulnerability life cycle-based security modeling and evaluation approach. *The Computer Journal* 56, 4 (2013), 422–439.

[253] Marczak, W. R., ET AL. When Governments Hack Opponents: A Look at Actors and Technology. In: tex.ids: marczak2014a, marczakWhenGovernmentsHacka, 16.

[254] McDermott, J. P. Attack net penetration testing. In: *Proceedings of the 2000 workshop on New security paradigms - NSPW '00*. ACM Press, 2000, 15–21. ISBN: 978-1-58113-260-1. DOI: 10.1145/366173.366183. URL: http://portal.acm.org/citation.cfm?doid=366173.366183.

[255] McHugh, J. Quality of protection: measuring the unmeasurable? In: *Proceedings of the 2nd ACM workshop on Quality of protection - QoP '06*. tex.ids: mchugh2006a tex.citation-number: 145. ACM Press, 2006, 1–2. ISBN: 978-1-59593-553-3. DOI: 10.1145/1179494.1179495. URL: http://portal.acm.org/citation.cfm?doid=1179494.1179495.

[256] McQueen, M., ET AL. Time-to- compromise model for cyber risk reduction estimation. In: *Quality of protection workshop at ESORICS 2005, the flagship european symposium on research in computer security*. Ed. by Gollmann, D., Massacci, F., AND Yautsiukhin, A. Vol. 23. Citation Key: mcqueen2005a tex.citation-number: 8. Springer US, 2005, 49–64. ISBN: 978-0-387-29016-4. DOI: 10.1007/978-0-387-36584-8_5. URL: http://link.springer.com/10.1007/978-0-387-36584-8%5C_5.

[257] Medina, A., ET AL. BRITE: An approach to universal topology generation. In: *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 2001, 346–353.

[258] Mehta, V., ET AL. Ranking attack graphs. In: *International Workshop on Recent Advances in Intrusion Detection*. tex.ids: mehta2006a tex.citation-number: 98. Springer, 2006, 127–144.

[259] Mell, P., Scarfone, K., AND Romanosky, S. A Complete Guide to the Common Vulnerability Scoring System Version 2.0 (). tex.ids: mell2007a, mellCompleteGuideCommona tex.citation-number: 47, 24.

[260] Mell, P., Scarfone, K., AND Romanosky, S. Common Vulnerability Scoring System. *IEEE Security and Privacy Magazine* 4, 6 (Nov. 2006). tex.ids: mell2006a tex.citation-number: 27, 85–89. ISSN: 1540-7993. DOI: 10.1109/MSP.2006.145.

[261] Mell, P., Scarfone, K., AND Romanosky, S. *The common vulnerability scoring system (CVSS) and its applicability to federal agency systems*. NIST IR 7435. 2007, NIST IR 7435. DOI: 10.6028/NIST.IR.7435. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7435.pdf.

[262] Mell, P., ET AL. *The Common Vulnerability Scoring System (CVSS) and Its Applicability to Federal Agency Systems*. 2007.

[263] Mellado, D., Fernández-Medina, E., AND Piattini, M. A comparison of software design security metrics. In: *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ECSA '10. tex.ids: d2010a tex.citation-number: 23. Association for Computing Machinery, Aug. 2010, 236–242. ISBN: 978-1-4503-0179-4. DOI: 10.1145/1842752.1842797. URL: https://doi.org/10.1145/1842752.1842797.

[264] Meneely, A., AND Williams, L. Strengthening the empirical analysis of the relationship between Linus' Law and software security. In: *Proceedings of the 2010*

ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. 2010, 1–10.

[265] Merkow, M., AND Breithaupt, J. *Computer security assurance, using the common criteria*. Citation Key: merkow2005a tex.citation-number: 42. Thomson Delmar Learning, 2005.

[266] Mezzour, G., Carley, K. M., AND Carley, L. R. An empirical study of global malware encounters. In: *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security - HotSoS '15*. tex.ids: mezzour2015a, mezzourEmpiricalStudyGlobal2015a. ACM Press, 2015, 1–11. ISBN: 978-1-4503-3376-4. DOI: 10.1145/2746194.2746202. URL: http://dl.acm.org/citation.cfm?doid=2746194.2746202.

[267] Michael, AND Williams, L. Toward the Use of Automated Static Analysis Alerts for Early Identification of Vulnerability- and Attack-prone Components. In: *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*. IEEE, July 2007, 18–18. DOI: 10.1109/ICIMP.2007.46. URL: https://ieeexplore.ieee.org/document/4271764/.

[268] Microsoft Security intelligence report (2013). Citation Key: microsoft2013a. URL: http://www.microsoft.com/security/sir/.

[269] Mihajlovic, V., AND Petkovic, M. *Dynamic bayesian networks: A state of the art, TR-CTI*. Citation Key: mihajlovic2001a tex.citation-number: 103. 2001.

[270] Milenkoski, A., ET AL. Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices. *ACM Computing Surveys* 48, 1 (Sept. 2015). tex.ids: milenkoski2015a, 1–41. ISSN: 03600300. DOI: 10.1145/2808691.

[271] Mir, I. A. Modeling of Security Measurement Metrics in an Information System (2013).

[272] Mohaisen, A., AND Alrawi, O. AV-Meter: An Evaluation of Antivirus Scans and Labels. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Ed. by Dietrich, S. Vol. 8550. tex.ids: mohaisen2014a, mohaisenAVMeterEvaluationAntivirus2014a. Springer International Publishing, 2014, 112–131. ISBN: 978-3-319-08508-1. DOI: 10.1007/978-3-319-08509-8_7. URL: http://link.springer.com/10.1007/978-3-319-08509-8%5C_7.

[273] Mohamed, W. N. H. W., Salleh, M. N. M., AND Omar, A. H. A comparative study of Reduced Error Pruning method in decision tree algorithms. In: *2012 IEEE International Conference on Control System, Computing and Engineering*. Nov. 2012, 392–397. DOI: 10.1109/ICCSCE.2012.6487177.

[274] Morales, J., Xu, S., AND Sandhu, R. Analyzing malware detection eciency with multiple anti-malware programs. *ASE Sci. J* 1, 2 (2012). Citation Key: morales2012a.

[275] Morana, M. M., AND Uceda Vélez, T. *Risk centric threat modeling: process for attack simulation and threat analysis*. Wiley, 2015. ISBN: 978-1-118-98835-0.

[276] Morris, A. S. *Measurement and instrumentation principles*. Butterworth-Heinemann, 2001. ISBN: 978-0-7506-5081-6.

[277] Morrison, P., ET AL. Mapping the field of software life cycle security metrics. *Information and Software Technology* 102 (Oct. 2018), 146–159. ISSN: 09505849. DOI: 10.1016/j.infsof.2018.05.011.

[278] Moses, T. Web services security quality of protection (2002). Citation Key: moses2002a tex.citation-number: 61. URL: http://xml.coverpages.org/ni2002-09-21-a.html.

[279] MP Azuwa, A., Ahmad, R., AND Sahib, S. Technical security metrics model in compliance with ISO/IEC 27001 standard. *International Journal of Cyber-Security and Digital Forensics* 1 (2012), 280–288.

[280] Musman, S., AND Turner, A. A game theoretic approach to cyber security risk management. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology* 15, 2 (Apr. 2018), 127–146. ISSN: 1548-5129, 1557-380X. DOI: 10.1177/1548512917699724.

[281] Muthukrishnan, K., AND Malis, A. *A Core MPLS IP VPN Architecture*. URL: https://tools.ietf.org/html/rfc2917.

[282] Nappa, A., ET AL. The Attack of the Clones: A Study of the Impact of Shared Code on Vulnerability Patching. In: *2015 IEEE Symposium on Security and Privacy*. tex.ids: nappa2015a. May 2015, 692–708. DOI: 10.1109/SP.2015.48.

[283] Nayak, K., ET AL. Some Vulnerabilities Are Different Than Others. In: vol. 8688. tex.ids: nayak2014a. 2014, 426–446. DOI: 10.1007/978-3-319-11379-1_21. URL: http://link.springer.com/10.1007/978-3-319-11379-1%5C_21.

[284] Nessus The nessus security scanner (). Citation Key: nessus-a tex.citation-number: 85 tex.type: [Online]. URL: http://www.nessus.org.

[285] Neupane, A., ET AL. A Multi-Modal Neuro-Physiological Study of Phishing Detection and Malware Warnings. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. tex.ids: neupane2015a. ACM Press, 2015, 479–491. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813660. URL: http://dl.acm.org/citation.cfm?doid=2810103.2813660.

[286] Nicol, D., Sanders, W., AND Trivedi, K. Model-based evaluation: from dependability to security. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (Jan. 2004). tex.ids: nicol2004a, nicolModelbasedEvaluationDependability2004a tex.citation-number: 52, 48–65. ISSN: 2160-9209. DOI: 10.1109/TDSC.2004.11.

[287] Nicol, D., ET AL. The science of security 5 hard problems (2015). Citation Key: nicol2015a. URL: http://cps-vo.org/node/21590..

[288] Niu, B., AND Tan, G. Per-Input Control-Flow Integrity. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. tex.ids: niu2015a. ACM Press, 2015, 914–926. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813644. URL: http://dl.acm.org/citation.cfm?doid=2810103.2813644.

[289] Noel, S. Text Mining for Modeling Cyberattacks. In: *Handbook of Statistics*. Jan. 2018. DOI: 10.1016/bs.host.2018.06.001.

[290] Noel, S., AND Jajodia, S. Managing attack graph complexity through visual hierarchical aggregation. In: *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*. tex.ids: noel2004a tex.citation-number: 87. ACM, 2004, 109–118. URL: http://dl.acm.org/citation.cfm?id=1029225.

[291] Noel, S., AND Jajodia, S. Measuring Security Risk of Networks Using Attack Graphs. 1, 1 (). tex.ids: noel2010a tex.citation-number: 99, 13.

[292] Noel, S., AND Jajodia, S. Metrics suite for network attack graph analytics. In: *Proceedings of the 9th Annual Cyber and Information Security Research Conference on - CISR '14*. tex.ids: noel2014a, noelMetricsSuiteNetwork2014a. ACM Press, 2014, 5–8. ISBN: 978-1-4503-2812-8. DOI: 10.1145/2602087.2602117. URL: http://dl.acm.org/citation.cfm?doid=2602087.2602117.

[293] Oltramari, A., ET AL. Towards a Human Factors Ontology for Cyber Security (), 8.

[294] Ong, C., Nahrstedt, K., AND Yuan, W. Quality of protection for mobile multimedia applications. In: *Multimedia and expo, 2003. ICME '03. Proceedings. 2003 international conference on*. Vol. 2. Citation Key: ong2003a tex.citation-number: 57. July 2003, 137–40.

[295] Oppenheimer, D., ET AL. Practical Issues in Dependability Benchmarking (), 6.

[296] Oroojlooyjadid, A., ET AL. A Deep Q-Network for the Beer Game: A Deep Reinforcement Learning algorithm to Solve Inventory Optimization Problems.

arXiv:1708.05924 [cs] (Feb. 2019). arXiv: 1708.05924. URL: http://arxiv.org/abs/1708.05924.

[297] Ortalo, R., Deswarte, Y., AND Kaâniche, M. Experimenting with quantitative evaluation tools for monitoring operational security. *IEEE Transactions on Software Engineering* 25, 5 (1999). tex.ids: ortalo1999a, ortaloExperimentingQuantitativeEvaluation1999a, 633–650.

[298] Ou, M., ET AL. Asymmetric transitivity preserving graph embedding. In: *Proceedings of the 22nd ACM SIGKDD*. ACM. 2016, 1105–1114.

[299] Ou, X., AND Appel, A. W. *A logic-programming approach to network security analysis*. Princeton University Princeton, 2005.

[300] Ou, X., Boyer, W. F., AND McQueen, M. A. A scalable approach to attack graph generation. In: *Proceedings of the 13th ACM conference on Computer and communications security*. ACM, 2006, 336–345. URL: http://dl.acm.org/citation.cfm?id=1180446.

[301] Ou, X., Govindavajhala, S., AND Appel, A. W. MulVAL: A Logic-based Network Security Analyzer. 14 (2005). Citation Key: ou2005a tex.citation-number: 84, 16.

[302] Ou, X., AND Singhal, A. *Quantitative Security Risk Assessment of Enterprise Networks*. SpringerBriefs in Computer Science. tex.ids: ou2011a. Springer New York, 2011. ISBN: 978-1-4614-1859-7. DOI: 10.1007/978-1-4614-1860-3. URL: http://link.springer.com/10.1007/978-1-4614-1860-3.

[303] Ouchani, S., AND Debbabi, M. Specification, verification, and quantifi- cation of security in model-based systems. *Computing* 97, 7 (July 2015). Citation Key: ouchani2015a tex.citation-number: 25, 691–711.

[304] Pamula, J., ET AL. A weakest-adversary security metric for network configuration security analysis. In: *Proceedings of the 2nd ACM workshop on Quality of protection - QoP '06*. tex.ids: pamula2006a, pamulaWeakestadversarySecurityMetric2006a tex.citation-number: 96. ACM Press, 2006, 31. ISBN: 978-1-59593-553-3. DOI:

10.1145/1179494.1179502. URL:
http://portal.acm.org/citation.cfm?doid=1179494.1179502.

[305] Paulauskas, N., AND Garsva, E. Attacker skill level distribution es- timation in the system mean time-to-compromise. In: *Information technology, 2008. IT 2008. 1st international conference on*. Citation Key: paulauskas2008a tex.citation-number: 77. May 2008, 1–4.

[306] Paxson, V. Strategies for sound internet measurement. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*. tex.ids: paxsonStrategiesSoundInternet2004a. ACM Press, 2004, 263. ISBN: 978-1-58113-821-4. DOI: 10.1145/1028788.1028824. URL:
http://portal.acm.org/citation.cfm?doid=1028788.1028824.

[307] Payne, S. *A Guide to Security Metrics*. Citation Key: payne2006a tex.citation-number: 22. 2006, 11.

[308] Pendleton, M., Garcia-Lebron, R., AND Xu, S. A Survey on Security Metrics. *arXiv:1601.05792 [cs]* (Jan. 2016). arXiv: 1601.05792. URL:
http://arxiv.org/abs/1601.05792.

[309] Pendleton, M., ET AL. A Survey on Systems Security Metrics. *ACM Computing Surveys* 49, 4 (Dec. 2016). tex.ids: pendleton2016a, pendletonSurveySystemsSecurity2016a, pendletonSurveySystemsSecurity2016b, pendletonSurveySystemsSecurity2016c, pendletonSurveySystemsSecurity2016d, pendletonSurveySystemsSecurity2016e, pendletonSurveySystemsSecurity2016f, pendletonSurveySystemsSecurity2016g tex.citation-number: 141, 1–35. ISSN: 03600300. DOI: 10.1145/3005714.

[310] Perl, H., ET AL. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, 426–437.

[311] Perozzi, B., Al-Rfou, R., AND Skiena, S. Deepwalk: Online learning of social representations. In: *Proceedings of KDD*. 2014, 701–710.

[312] Pfleeger, S. L. Useful Cybersecurity Metrics. *IT Professional Magazine; Washington* 11, 3 (June 2009). tex.ids: pfleeger2009a, 38–45. ISSN: 15209202. DOI: http://dx.doi.org/10.1109/MITP.2009.63.

[313] Pfleeger, S., AND Cunningham, R. Why Measuring Security Is Hard. *IEEE Security Privacy* 8, 4 (July 2010). tex.ids: pfleeger2010a tex.citation-number: 140, 46–54. ISSN: 1558-4046. DOI: 10.1109/MSP.2010.60.

[314] Phanekham, D., Zaber, M., AND Nair, S. *Measuring Cloud Network Performance with PerfKit Benchmarker*.

[315] Phillips, C., AND Swiler, L. P. A graph-based system for network-vulnerability analysis. In: *Proceedings of the 1998 workshop on New security paradigms - NSPW '98*. tex.ids: phillips1998a, phillipsGraphbasedSystemNetworkvulnerability1998a tex.citation-number: 83. ACM Press, 1998, 71–79. ISBN: 978-1-58113-168-0. DOI: 10.1145/310889.310919. URL: http://portal.acm.org/citation.cfm?doid=310889.310919.

[316] Piessens, F. The cyber security body of knowledge. In: tex.chapter: Software Security. University of Bristol, 2019. URL: https://www.cybok.org/.

[317] Plummer, D. *An Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*. URL: https://tools.ietf.org/html/rfc826.

[318] Poolsappasit, N., Dewri, R., AND Ray, I. Dynamic security risk management using bayesian attack graphs. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (Jan. 2012). Citation Key: poolsappasit2012a tex.citation-number: 107, 61–74.

[319] Prakash, A., AND Wellman, M. P. Empirical Game-Theoretic Analysis for Moving Target Defense. In: *Proceedings of the Second ACM Workshop on Moving Target Defense - MTD '15*. tex.ids: prakash2015a. ACM Press, 2015, 57–65. ISBN:

978-1-4503-3823-3. DOI: 10.1145/2808475.2808483. URL: http://dl.acm.org/citation.cfm?doid=2808475.2808483.

[320] Premaratne, U., ET AL. Security Analysis and Auditing of IEC61850-Based Automated Substations. *IEEE Transactions on Power Delivery* 25, 4 (Oct. 2010). tex.ids: premaratne2010a tex.citation-number: 4, 2346–2355. ISSN: 1937-4208. DOI: 10.1109/TPWRD.2010.2043122.

[321] Qiu, J., ET AL. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining - WSDM '18* (2018). arXiv: 1710.02971, 459–467. DOI: 10.1145/3159652.3159706.

[322] Qu, M., Bengio, Y., AND Tang, J. GMNN: Graph Markov Neural Networks (), 10.

[323] Rajab, M., Monrose, F., AND Terzis, A. On the effectiveness of distributed worm monitoring. In: *Proc. USENIX security symposium. Google scholar*. Citation Key: rajab2005a. 2005.

[324] Ramos, A., AND Filho, R. Sensor data security level estimation scheme for wireless sensor networks. *Sensors* 15, 1 (Jan. 2015). Citation Key: ramos2015a tex.citation-number: 33, 2104–2136.

[325] Ramos, A., ET AL. A security metric for the evaluation of collaborative intrusion de- tection systems in wireless sensor networks. In: *IEEE international conference on communications*. Citation Key: ramos-a tex.citation-number: 34.

[326] Ramos, A., ET AL. Quantifying node security in wireless sensor networks under worm attacks. In: *Brazilian symposium of computer networks*. Citation Key: ramos-b tex.citation-number: 35.

[327] Ramos, A., ET AL. Model-Based Quantitative Network Security Metrics: A Survey. *IEEE Communications Surveys Tutorials* 19, 4 (2017), 2704–2734. ISSN: 2373-745X. DOI: 10.1109/COMST.2017.2745505.

[328] Rao, P., ET AL. XSB: A system for efficiently computing well-founded semantics. In: *Logic Programming And Nonmonotonic Reasoning*. Ed. by Dix, J., Furbach, U., AND Nerode, A. Springer Berlin Heidelberg, 1997, 430–440. ISBN: 978-3-540-69249-2.

[329] Rashid, A., ET AL. The Cyber Security Body of Knowledge (). tex.ids: rashidCyberSecurityBodya, rashidCyberSecurityBodyb, 854.

[330] Reiter, M. K., AND Stubblebine, S. G. Authentication metric analysis and design. *ACM Transactions on Information and System Security* 2, 2 (May 1999). tex.ids: reiter1999a, reiterAuthenticationMetricAnalysis1999a, 138–158. ISSN: 10949224. DOI: 10.1145/317087.317088.

[331] Rimal, B., Choi, E., AND Lumb, I. A taxonomy and survey of cloud computing systems. In: *INC, IMS and IDC, 2009. NCM '09. Fifth international joint conference on*. Citation Key: rimal2009a tex.citation-number: 130. Aug. 2009, 44–51.

[332] Ring, M., ET AL. Flow-based benchmark data sets for intrusion detection (). tex.ids: ringFlowbasedBenchmarkDataa, 10.

[333] Ritchey, R., AND Ammann, P. Using model checking to analyze network vulnerabilities. In: *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. tex.ids: ritchey2000a, ritcheyUsingModelChecking2000a tex.citation-number: 90 ISSN: 1081-6011. IEEE Comput. Soc, 2000, 156–165. ISBN: 978-0-7695-0665-4. DOI: 10.1109/SECPRI.2000.848453. URL: http://ieeexplore.ieee.org/document/848453/.

[334] rndic, N., AND Laskov, P. Practical Evasion of a Learning-Based Classifier: A Case Study. In: *2014 IEEE Symposium on Security and Privacy*. tex.ids: rndic2014a. IEEE, May 2014, 197–211. ISBN: 978-1-4799-4686-0. DOI: 10.1109/SP.2014.20. URL: http://ieeexplore.ieee.org/document/6956565/.

[335] Roberts, F. *Measurement theory, with applications to decision making, utility and the social sciences*. Citation Key: roberts1979a. Addison-Wesley, Boston.Google Scholar, 1979.

[336] Rosen, E. C., AND Rekhter, Y. BGP/MPLS IP virtual private networks (VPNs) (2006).

[337] Rosenquist, M. Prioritizing Information Security Risks with Threat Agent Risk Assessment (), 8.

[338] Rostami, M., ET AL. Hardware security: Threat models and metrics. In: *Proc. Int. Conf.* Citation Key: rostami2013a tex.citation-number: 13. Computer Aided Design, 2013, 819–823.

[339] Rostami, M., Koushanfar, F., AND Karri, R. A Primer on Hardware Security: Models, Methods, and Metrics. *Proceedings of the IEEE* 102, 8 (Aug. 2014). tex.ids: rostamiPrimerHardwareSecurity2014a, 1283–1295. ISSN: 1558-2256. DOI: 10.1109/JPROC.2014.2335155.

[340] Roundy, K. A., AND Miller, B. P. Binary-code obfuscations in prevalent packer tools. *ACM Computing Surveys* 46, 1 (Oct. 2013). tex.ids: roundy2013a, 1–32. ISSN: 03600300. DOI: 10.1145/2522968.2522972.

[341] Roussev, V. The cyber security body of knowledge. In: tex.chapter: Forensics. University of Bristol, 2019. URL: https://www.cybok.org/.

[342] Rudolph, M., AND Schwarz, R. Security indicators–a state of the art survey public report. *FhG IESE VII (043)* (Aug. 2012). tex.ids: rudolph2012a, rudolphCriticalSurveySecurity2012a tex.citation-number: 17 ISSN: null, 291–300. DOI: 10.1109/ARES.2012.10.

[343] Sabottke, C., Suciu, O.-P., AND Dumitras, T. Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits. In: tex.ids: sabottke2015a, 16.

[344] Sahinoglu, M. An input-output measurable design for the security meter model to quantify and manage software security risk. *IEEE Transactions on Instrumentation and Measurement* 57, 6 (2008). Citation Key: sahinoglu2008a tex.citation-number: 123, 1251–1260.

[345] Sahinoglu, M. Security meter: A practical decision-tree model to quantify risk. *IEEE Security and Privacy* 3, 3 (2005). Citation Key: sahinoglu2005a tex.citation-number: 122, 18–24.

[346] Saitta, P., Larcom, B., AND Eddington, M. Trike v.1 Methodology Document [Draft] (), 17.

[347] Sallhammar, K. Towards a stochastic model for integrated security and dependability evaluation. In: *First international conference on availability, reliability and security (ARES'06*. Citation Key: sallhammar2006b tex.citation-number: 54. Apr. 2006, 8.

[348] Sallhammar, K., Helvik, B., AND Knapskog, S. On stochastic mod- eling for integrated security and dependability evaluation. *Journal of Networks* 1, 5 (Oct. 2006). Citation Key: sallhammar2006a tex.citation-number: 53.

[349] Sanders, W. H. Quantitative Security Metrics: Unattainable Holy Grail or a Vital Breakthrough within Our Reach? *IEEE Security Privacy* 12, 2 (Mar. 2014). tex.ids: sanders2014a, 67–69. ISSN: 1558-4046. DOI: 10.1109/MSP.2014.31.

[350] Sasse, M. A. The cyber security body of knowledge. In: tex.chapter: Human Factors. University of Bristol, 2019. URL: https://www.cybok.org/.

[351] Savola, R. Towards a Security Metrics Taxonomy for the Information and Communication Technology Industry. In: *International Conference on Software Engineering Advances (ICSEA 2007)*. tex.ids: savola2007a tex.citation-number: 15 ISSN: null. Aug. 2007, 60–60. DOI: 10.1109/ICSEA.2007.79.

[352] Savola, R. M. Quality of security metrics and measurements. *Computers & Security* 37 (Sept. 2013), 78–90. ISSN: 0167-4048. DOI: 10.1016/j.cose.2013.05.002.

[353] Scarfone, K., AND Mell, P. An analysis of cvss version 2 vulnerability scoring. In: *2009 3rd international symposium on empirical software engineering and measurement*. Citation Key: scarfone2009a tex.citation-number: 48. Oct. 2009, 516–525.

[354] Scarfone, K., AND Mell, P. Vulnerability scoring for security configuration settings. In: *Proceedings of the 4th ACM workshop on Quality of protection*. 2008, 3–8.

[355] Schneider, F. B. Blueprint for a Science of Cybersecurity (), 17.

[356] Schneier, B. Secrets 8 lies: Digital security in a networked world (2000). Citation Key: schneier2000a.

[357] Schneier, B. Attack trees. *Dr. Dobb's journal* 24, 12 (1999). tex.ids: schneierAttackTrees1999a, 21–29.

[358] Schoenfield, B. S. E. Threat Modeling Demystified (), 98.

[359] Schryen, G., AND Kadura, R. Open source vs. closed source software: towards measuring security. In: *Proceedings of the 2009 ACM symposium on Applied Computing*. 2009, 2016–2023.

[360] Science, C., ET AL. *Implications of Artificial Intelligence for Cybersecurity: Proceedings of a Workshop*. Ed. by Johnson, A., AND Grumbling, E. National Academies Press, Dec. 2019. ISBN: 978-0-309-49450-2. DOI: 10.17226/25488. URL: https://www.nap.edu/catalog/25488.

[361] Sciences, N. A. o., Engineering, AND Medicine *Implications of artificial intelligence for cybersecurity: Proceedings of a workshop*. Ed. by Johnson, A., AND Grumbling, E. Citation Key: NAP25488. The National Academies Press, 2019. ISBN: 978-0-309-49450-2. DOI: 10.17226/25488. URL: https://www.nap.edu/catalog/25488/implications-of-artificial-intelligence-for-cybersecurity-proceedings-of-a-workshop.

[362] Sembiring, J., ET AL. Network Security Risk Analysis using Improved MulVAL Bayesian Attack Graphs. *International Journal on Electrical Engineering and Informatics* 7 (Dec. 2015), 735–753. DOI: 10.15676/ijeei.2015.7.4.15.

[363] Shacham, H., ET AL. On the effectiveness of address-space randomization. In: *Proceedings of the 11th ACM conference on Computer and communications security - CCS '04*. tex.ids: shacham2004a. ACM Press, 2004, 298. ISBN: 978-1-58113-961-7. DOI: 10.1145/1030083.1030124. URL: http://portal.acm.org/citation.cfm?doid=1030083.1030124.

[364] Al-Shaer, E., Khan, L., AND Ahmed, M. A comprehensive objective network security metric framework for proactive security configuration. *Proc. CSIIRW'08* 42, 1 (2008). Citation Key: al-shaer2008a tex.address: Google Scholar.

[365] Shandilya, V., Simmons, C., AND Shiva, S. Use of attack graphs in security systems. *Journal of Computer Networks and Communi- cations* 2014 (2014). Citation Key: shandilya2014a tex.citation-number: 93, 1–13.

[366] Sharafaldin, I., ET AL. Towards a Reliable Intrusion Detection Benchmark Dataset. *Software Networking* 2018, 1 (Jan. 2018), 177–200. ISSN: 2445-9739. DOI: 10.13052/jsn2445-9739.2017.009.

[367] Sheng, S., ET AL. Who falls for phish?: a demographic analysis of phishing susceptibility and effectiveness of interventions. In: *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. tex.ids: sheng2010a. ACM Press, 2010, 373. ISBN: 978-1-60558-929-9. DOI: 10.1145/1753326.1753383. URL: http://portal.acm.org/citation.cfm?doid=1753326.1753383.

[368] Shevchenko, N., ET AL. Threat Modeling: A Summary of Available Methods (), 26.

[369] Sheyner, O., ET AL. Automated generation and analysis of attack graphs. In: *Proceedings 2002 IEEE Symposium on Security and Privacy*. tex.ids: sheyner2002a, sheynerAutomatedGenerationAnalysis2002a. IEEE, 2002, 273–284. URL: http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=1004377.

[370] Shiravi, H., Shiravi, A., AND Ghorbani, A. A survey of visualization systems for network security. *IEEE Transactions on Visualization and Computer Graphics* 18, 8 (Aug. 2012). Citation Key: shiravi2012a tex.citation-number: 65, 1313–1329.

[371] Shostack, A. Experiences Threat Modeling at Microsoft (), 11.

[372] Singhal, A., AND Ou, X. Security Risk Analysis of Enterprise Networks Using Probabilistic Attack Graphs (). tex.ids: singhal2011a, singhalSecurityRiskAnalysis2017 tex.publisher: National Institute of Standards and Technology, 24.

[373] *Single Metric for HPC (NERSC)*. URL: https://www.nersc.gov/research-and-development/benchmarking-and-workload-characterization/ssi/.

[374] Smart, N. The cyber security body of knowledge. In: tex.chapter: Cryptography. University of Bristol, 2019. URL: https://www.cybok.org/.

[375] Smith, J. E. Characterizing computer performance with a single number. *Communications of the ACM* 31, 10 (1988), 1202–1206.

[376] Snoek, J., Larochelle, H., AND Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. *arXiv:1206.2944 [cs, stat]* (June 2012). arXiv: 1206.2944. URL: http://arxiv.org/abs/1206.2944.

[377] Snoek, J., ET AL. Scalable Bayesian Optimization Using Deep Neural Networks. *arXiv:1502.05700 [stat]* (Feb. 2015). arXiv: 1502.05700. URL: http://arxiv.org/abs/1502.05700.

[378] Snow, K. Z., ET AL. Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization. In: *2013 IEEE Symposium on Security and Privacy*. tex.ids: snow2013a. May 2013, 574–588. DOI: 10.1109/SP.2013.45.

[379] Solic, K., Ocevcic, H., AND Golub, M. The information systems' security level assessment model based on an ontology and evidential reasoning approach. *Computers & Security* 55 (Nov. 2015). Citation Key: solic2015a tex.citation-number: 43, 100–112.

[380] Sönmez, F. Ö. A Conceptual Model for a Metric Based Framework for the Monitoring of Information Security Tasks' Efficiency. *Procedia Computer Science* 160 (2019), 181–188. ISSN: 18770509. DOI: 10.1016/j.procs.2019.09.459.

[381] Spring, J. M., Moore, T., AND Pym, D. Practicing a Science of Security: A Philosophy of Science Perspective. In: *Proceedings of the 2017 New Security Paradigms Workshop on ZZZ - NSPW 2017*. ACM Press, 2017, 1–18. ISBN: 978-1-4503-6384-6. DOI: 10.1145/3171533.3171540. URL: http://dl.acm.org/citation.cfm?doid=3171533.3171540.

[382] Srivatanakul, T., Clark, J. A., AND Polack, F. Effective Security Requirements Analysis: HAZOP and Use Cases. In: *Information Security*. Ed. by Zhang, K., AND Zheng, Y. Vol. 3225. Springer Berlin Heidelberg, 2004, 416–427. ISBN: 978-3-540-23208-7. DOI: 10.1007/978-3-540-30144-8_35. URL: http://link.springer.com/10.1007/978-3-540-30144-8%5C_35.

[383] Stakhanova, N., ET AL. Towards cost-sensitive assessment of intrusion response selection. *Journal of Computer Security* 20, 2–3 (June 2012). tex.ids: stakhanova2012a, 169–198. ISSN: 18758924, 0926227X. DOI: 10.3233/JCS-2011-0436.

[384] Stan, O., ET AL. Extending Attack Graphs to Represent Cyber-Attacks in Communication Protocols and Modern IT Networks. *arXiv:1906.09786 [cs]* (June 2019). arXiv: 1906.09786. URL: http://arxiv.org/abs/1906.09786.

[385] *STATE OF SOFTWARE SECURITY*. URL: https://www.veracode.com/sites/default/files/pdf/resources/ipapers/state-of-software-security-volume-9/.

[386] Stevens, S. S. On the Theory of Scales of Measurement. *Science, New Series* 103, 2684 (1946). tex.ids: stevens1946a, 677–680.

[387] Stolfo, S., Bellovin, S., AND Evans, D. Measuring security. *IEEE Security and Privacy* 9, 3 (May 2011). Citation Key: stolfo2011a tex.citation-number: 12, 60–65.

[388] Stolfo, S., ET AL. Cost-based modeling for fraud and intrusion detection: results from the JAM project. In: *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*. Vol. 2. tex.ids: stolfo2000a. IEEE Comput. Soc, 1999, 130–144. ISBN: 978-0-7695-0490-2. DOI: 10.1109/DISCEX.2000.821515. URL: http://ieeexplore.ieee.org/document/821515/.

[389] Stone-Gross, B., ET AL. FIRE: FInding Rogue nEtworks. In: *2009 Annual Computer Security Applications Conference*. tex.ids: stone-gross2009a. IEEE, Dec. 2009, 231–240. ISBN: 978-1-4244-5327-6. DOI: 10.1109/ACSAC.2009.29. URL: http://ieeexplore.ieee.org/document/5380682/.

[390] Strasburg, C., ET AL. A Framework for Cost Sensitive Assessment of Intrusion Response Selection. In: *2009 33rd Annual IEEE International Computer Software and Applications Conference*. tex.ids: strasburg2009a. IEEE, 2009, 355–360. ISBN: 978-0-7695-3726-9. DOI: 10.1109/COMPSAC.2009.54. URL: http://ieeexplore.ieee.org/document/5254241/.

[391] Stringhini, G. The cyber security body of knowledge. In: tex.chapter: Adversarial Behaviours. University of Bristol, 2019. URL: https://www.cybok.org/.

[392] Sullivan, L. H. Chapter 9: Dataflow Diagrams (), 50.

[393] Suri, N. The cyber security body of knowledge. In: tex.chapter: Distributed Systems Security. University of Bristol, 2019. URL: https://www.cybok.org/.

[394] Survey and Benchmark of Lightweight Block Ciphers for Wireless Sensor Networks. In: *Proceedings of the 10th International Conference on Security and Cryptography*. SCITEPRESS - Science, 2013, 543–548. ISBN: 978-989-8565-73-0. DOI:

10.5220/0004530905430548. URL: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0004530905430548.

[395] Sutton, R. S., AND Barto, A. G. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. The MIT Press, 2018. ISBN: 978-0-262-03924-6.

[396] Swanson, M., ET AL. *Security metrics guide for information technology systems*. NIST SP 800-55. 2003, NIST SP 800–55. DOI: 10.6028/NIST.SP.800-55. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-55.pdf.

[397] Tang, A., Sethumadhavan, S., AND Stolfo, S. Heisenbyte: Thwarting Memory Disclosure Attacks using Destructive Code Reads. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*. tex.ids: tang2015a. ACM Press, 2015, 256–267. ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813685. URL: http://dl.acm.org/citation.cfm?doid=2810103.2813685.

[398] Tang, J., ET AL. Line: Large-scale information network embedding. In: *Proceedings of WWW*. 2015, 1067–1077.

[399] Tang, T. A., ET AL. Deep learning approach for Network Intrusion Detection in Software Defined Networking. In: *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*. IEEE, Oct. 2016, 258–263. ISBN: 978-1-5090-3837-4. DOI: 10.1109/WINCOM.2016.7777224. URL: http://ieeexplore.ieee.org/document/7777224/.

[400] Tariq, M. I. Towards information security metrics framework for cloud computing. *International Journal of Cloud Computing and Services Science* 1, 4 (2012), 209.

[401] Tavallaee, M., Stakhanova, N., AND Ghorbani, A. A. Toward Credible Evaluation of Anomaly-Based Intrusion-Detection Methods. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40, 5 (Sept. 2010).

tex.ids: tavallaee2010a, 516–524. ISSN: 1094-6977, 1558-2442. DOI: 10.1109/TSMCC.2010.2048428.

[402] Thakore, U. A QUANTITATIVE METHODOLOGY FOR EVALUATING AND DEPLOYING SECURITY MONITORS (). tex.ids: thakore2015a, thakoreQUANTITATIVEMETHODOLOGYEVALUATINGa, 87.

[403] Trček, D. Security metrics foundations for computer security. *The Computer Journal* 53, 7 (2010), 1106–1112.

[404] Troncoso, C. The cyber security body of knowledge. In: tex.chapter: Privacy & Online Rights. University of Bristol, 2019. URL: https://www.cybok.org/.

[405] TU, C., ET AL. Network representation learning: an overview. *SCIENTIA SINICA Informationis* 47, 8 (2017), 980–996.

[406] Tupper, M., AND Zincir-Heywood, A. N. VEA-bility Security Metric: A Network Security Analysis Tool. In: *2008 Third International Conference on Availability, Reliability and Security*. tex.ids: tupper2008a, tupperVEAbilitySecurityMetric2008a tex.citation-number: 31. IEEE, Mar. 2008, 950–957. ISBN: 978-0-7695-3102-1. DOI: 10.1109/ARES.2008.138. URL: http://ieeexplore.ieee.org/document/4529446/.

[407] Ugarte-Pedrero, X., ET AL. SoK: Deep Packer Inspection: A Longitudinal Study of the Complexity of Run-Time Packers. In: *2015 IEEE Symposium on Security and Privacy*. tex.ids: ugarte-pedrero2015a, ugarte-pedreroSoKDeepPacker2015a, ugarte-pedreroSoKDeepPacker2015b. IEEE, May 2015, 659–673. ISBN: 978-1-4673-6949-7. DOI: 10.1109/SP.2015.46. URL: https://ieeexplore.ieee.org/document/7163053/.

[408] Ur Rahman, A. A., AND Williams, L. Security practices in DevOps. In: *Proceedings of the Symposium and Bootcamp on the Science of Security - HotSos '16*. ACM Press, 2016, 109–111. ISBN: 978-1-4503-4277-3. DOI: 10.1145/2898375.2898383. URL: http://dl.acm.org/citation.cfm?doid=2898375.2898383.

[409] Ur, B., ET AL. How Does Your Password Measure Up? The Effect of Strength Meters on Password Creation (). tex.ids: ur2012a, 16.

[410] Ur, B., ET AL. Measuring Real-World Accuracies and Biases in Modeling Password Guessability (). tex.ids: ur2015a, urMeasuringRealWorldAccuraciesa, 19.

[411] Vaughn, R., Henning, R., AND Siraj, A. Information assurance measures and metrics - state of practice and proposed taxonomy. In: *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the.* tex.ids: vaughn2003a tex.citation-number: 14. IEEE, 2003, 10 pp. ISBN: 978-0-7695-1874-9. DOI: 10.1109/HICSS.2003.1174904. URL: http://ieeexplore.ieee.org/document/1174904/.

[412] Vellaithurai, C., ET AL. Cpin- dex: Cyber-physical vulnerability assessment for power-grid infrastruc- tures. *IEEE Transactions on Smart Grid* 6, 2 (Mar. 2015). Citation Key: vellaithurai2015a tex.citation-number: 119, 566–575.

[413] Velleman, P., AND Wilkinson, L. Nominal, Ordinal, Interval, and Ratio Typologies are Misleading (). tex.ids: velleman1993a, 19.

[414] Verbauwhede, I. The cyber security body of knowledge. In: tex.chapter: Hardware Security. University of Bristol, 2019. URL: https://www.cybok.org/.

[415] Verendel, V. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In: *Proceedings of the 2009 workshop on New security paradigms workshop - NSPW '09.* tex.ids: verendel2009a, verendelQuantifiedSecurityWeak2009a, verendelQuantifiedSecurityWeak2009b tex.citation-number: 16. ACM Press, 2009, 37. ISBN: 978-1-60558-845-2. DOI: 10.1145/1719030.1719036. URL: http://portal.acm.org/citation.cfm?doid=1719030.1719036.

[416] Vigna, G., AND Kemmerer, R. Netstat: A network-based intrusion detection system. *J. Comput. Secur* 7, 1 (Jan. 1999). Citation Key: vigna1999a tex.citation-number: 86, 37–71.

[417] Villarrubia, C., Fernández-Medina, E., AND Piattini, M. Metrics of Password Management Policy. *Computational Science and Its Applications-ICCSA 2006* (2006), 1013–1023.

[418] Villarrubia, C., Medina, E. F., AND Piattini, M. Towards a classification of security metrics. In: *WOSIS*. Citation Key: villarrubia2004a. 2004, 342–350.

[419] Wagner, I., AND Eckhoff, D. *Technical privacy metrics: A systematic survey.* Vol. 51. 3. Citation Key: wagner2015a tex.arxiv: 1512.00327. June 2015, 1–38. DOI: 10.1145/3168389.

[420] Wald, A. A method of estimating plane vulnerability based on damage of survivors, CRC 432, July 1980. *Center for Naval Analyses* (1980).

[421] Wang, D., Cui, P., AND Zhu, W. Structural deep network embedding. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, 1225–1234.

[422] Wang, F., ET AL. Sitar: a scalable intrusion-tolerant architecture for dis- tributed services. In: *Foundations of intrusion tolerant systems, 2003 [Organically assured and survivable information systems*. Citation Key: wang2003a tex.citation-number: 133. 2003, 359–367.

[423] Wang, L., ET AL. Mod- eling network diversity for evaluating the robustness of networks against zero-day attacks. In: *Proceedings of the 19th european symposium on research in computer security*. Citation Key: wang2014b tex.citation-number: 116. Springer International Publishing, 2014, 494–511.

[424] Wang, L., Jajodia, S., AND Singhal, A. *Network Security Metrics*. Springer International Publishing, 2017. ISBN: 978-3-319-66504-7. DOI: 10.1007/978-3-319-66505-4. URL: http://link.springer.com/10.1007/978-3-319-66505-4.

[425] Wang, L., Singhal, A., AND Jajodia, S. Measuring the overall security of network configurations using attack graphs. In: *IFIP Annual Conference on Data and Applications Security and Privacy*. tex.ids: wang2007a tex.citation-number: 100. Springer, 2007, 98–112.

[426] Wang, L., Singhal, A., AND Jajodia, S. Toward Measuring Network Security Using Attack Graphs. In: tex.ids: wang2007a, wang2007b tex.citation-number: 101. 2007.

[427] Wang, L., ET AL. An attack graph-based probabilistic security metric. In: *IFIP Annual Conference on Data and Applications Security and Privacy*. tex.ids: wang2008a, wangAttackGraphbasedProbabilistic2008a tex.citation-number: 88. Springer, 2008, 283–296. URL: http://link.springer.com/chapter/10.1007/978-3-540-70567-3%5C_22.

[428] Wang, L., ET AL. k-Zero Day Safety: A Network Security Metric for Measuring the Risk of Unknown Vulnerabilities. *IEEE Transactions on Dependable and Secure Computing* 11, 1 (Jan. 2014). tex.ids: wang2014a, wangKzeroDaySafety2013 tex.citation-number: 113 publisher: IEEE, 30–44. ISSN: 1545-5971. DOI: 10.1109/TDSC.2013.24.

[429] Wang, L., ET AL. k-Zero Day Safety: Measuring the Security Risk of Networks against Unknown Attacks. In: *Computer Security – ESORICS 2010*. Ed. by Gritzalis, D., Preneel, B., AND Theoharidou, M. Vol. 6345. tex.ids: wang2010a. Springer Berlin Heidelberg, 2010, 573–587. ISBN: 978-3-642-15496-6. DOI: 10.1007/978-3-642-15497-3_35. URL: http://link.springer.com/10.1007/978-3-642-15497-3%5C_35.

[430] Wei, H., ET AL. Cost-benefit analysis for network intrusion detection systems. In: *Proceedings of the 28th annual computer security conference*. Citation Key: wei2001a. D.C.Google Scholar, 2001.

[431] Weir, M., ET AL. Testing metrics for password creation policies by attacking large sets of revealed passwords. In: *Proceedings of the 17th ACM conference on*

Computer and communications security - CCS '10. tex.ids: weir2010a, weirTestingMetricsPassword2010a. ACM Press, 2010, 162. ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866327. URL: http://portal.acm.org/citation.cfm?doid=1866307.1866327.

[432] Weiss, J. D. A system security engineering process. In: *Proceedings of the 14th National Computer Security Conference*. Vol. 249. 1991, 572–581.

[433] Whaiduzzaman, M., AND Gani, A. Measuring security for cloud service provider: A third party approach. In: *Electrical information and communication technology (EICT), 2013 international conference on*. Citation Key: whaiduzzaman2014a tex.citation-number: 36. Feb. 2014, 1–6.

[434] Williams, L. The cyber security body of knowledge. In: tex.chapter: Secure Software Lifecycle. University of Bristol, 2019. URL: https://www.cybok.org/.

[435] Woodard, L., ET AL. *Categorizing threat: building and using a generic threat matrix.* SAND2007-5791, 921121. Sept. 2007, SAND2007–5791, 921121. DOI: 10.2172/921121. URL: http://www.osti.gov/servlets/purl/921121-o2fi48/.

[436] Wynn, J., ET AL. Methodology Description Version 1.0 (), 60.

[437] Xie, A., ET AL. Applying attack graphs to network security metric. In: *2009 international conference on multimedia information networking and security*. Vol. 1. Citation Key: xie2009a tex.citation-number: 41. IEEE, 2009, 427–431.

[438] Xie, P., ET AL. Using bayesian networks for cyber security analysis. In: *Dependable systems and networks (DSN), 2010 IEEE/IFIP international conference on*. Citation Key: xie2010a tex.citation-number: 108. June 2010, 211–220.

[439] Xin, Y., ET AL. Machine Learning and Deep Learning Methods for Cybersecurity. *IEEE Access* 6 (2018). tex.ids: xinMachineLearningDeep2018a, 35365–35381. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2836950.

[440] Xu, L., ET AL. An Evasion and Counter-Evasion Study in Malicious Websites Detection. *arXiv:1408.1993 [cs]* (Aug. 2014). tex.ids: xu2014b, xuEvasionCounterevasionStudy2014 arXiv: 1408.1993. ISSN: null. URL: http://arxiv.org/abs/1408.1993.

[441] Xu, M., Da, G., AND Xu, S. Cyber Epidemic Models with Dependences. *Internet Mathematics* (). tex.ids: xu2015b, xuCyberEpidemicModelsa, 37.

[442] Xu, M., AND Xu, S. An Extended Stochastic Model for Quantitative Security Analysis of Networked Systems. *Internet Mathematics* 8, 3 (Aug. 2012). tex.ids: mxu2012a, xuExtendedStochasticModel2012a, xuExtendedStochasticModel2012b, 288–320. ISSN: 1542-7951, 1944-9488. DOI: 10.1080/15427951.2012.654480.

[443] Xu, S., Lu, W., AND Xu, L. Push- and pull-based epidemic spreading in arbitrary networks: Thresholds and deeper insights. *ACM TAAS* 7, 3, 1 (2012). Citation Key: xu2012b tex.address: Google Scholar.

[444] Xu, S., Lu, W., AND Zhan, Z. A stochastic model of multivirus dynamics. *IEEE Trans. Depend. Secure Comput* 9, 1 (2012). Citation Key: xu2012a, 30–45.

[445] Xu, S. Cybersecurity dynamics. *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security - HotSoS '14* (2014). tex.ids: xu2014a, xuCybersecurityDynamics2014a tex.source: Google Scholar, 1–2. DOI: 10.1145/2600176.2600190.

[446] Xu, S. Emergent behavior in cybersecurity. In: *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security - HotSoS '14*. tex.ids: xu2014c, xuEmergentBehaviorCybersecurity2014a tex.source: Google Scholar. ACM Press, 2014, 1–2. ISBN: 978-1-4503-2907-1. DOI: 10.1145/2600176.2600189. URL: http://dl.acm.org/citation.cfm?doid=2600176.2600189.

[447] Xu, S., Lu, W., AND Li, H. A Stochastic Model of Active Cyber Defense Dynamics. *Internet Mathematics* (). tex.ids: xu2015a, xuStochasticModelActivea, 48.

[448] Xu, S., ET AL. Adaptive Epidemic Dynamics in Networks: Thresholds and Control. *ACM Transactions on Autonomous and Adaptive Systems* 8, 4 (Jan. 2014). tex.ids: xu2014d, xuAdaptiveEpidemicDynamics2014a, 1–19. ISSN: 15564665. DOI: 10.1145/2555613.

[449] Yamin, M. M., Katt, B., AND Gkioulos, V. Cyber ranges and security testbeds: Scenarios, functions, tools and architecture. *Computers & Security* 88 (Jan. 2020), 101636. ISSN: 0167-4048. DOI: 10.1016/j.cose.2019.101636.

[450] Yang, C., ET AL. Network representation learning with rich text information. In: *Proceedings of IJCAI*. 2015.

[451] Yardon, D. Symantec develops new attack on cyberhacking (May 2014). Citation Key: yardon2014a. URL: http://www.wsj..

[452] Yen, T.-F., ET AL. An Epidemiological Study of Malware Encounters in a Large Enterprise. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security - CCS '14*. tex.ids: yen2014a, yenEpidemiologicalStudyMalware2014a. ACM Press, 2014, 1117–1130. ISBN: 978-1-4503-2957-6. DOI: 10.1145/2660267.2660330. URL: http://dl.acm.org/citation.cfm?doid=2660267.2660330.

[453] Yilek, S., ET AL. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In: *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference - IMC '09*. tex.ids: yilek2009a, yilekWhenPrivateKeys2009a. ACM Press, 2009, 15. ISBN: 978-1-60558-771-4. DOI: 10.1145/1644893.1644896. URL: http://portal.acm.org/citation.cfm?doid=1644893.1644896.

[454] Yin, X., ET AL. Visflowconnect: providing security situational awareness by visualizing network traffic flows. In: *Performance, computing, and communications, 2004 IEEE international conference on*. Citation Key: yin2004a tex.citation-number: 66. 2004, 601–607.

[455] Zaffarano, K., Taylor, J., AND Hamilton, S. A Quantitative Framework for Moving Target Defense Effectiveness Evaluation. In: *Proceedings of the Second ACM Workshop on Moving Target Defense - MTD '15*. tex.ids: zaffarano2015a, zaffaranoQuantitativeFrameworkMoving2015a. ACM Press, 2015, 3–10. ISBN: 978-1-4503-3823-3. DOI: 10.1145/2808475.2808476. URL: http://dl.acm.org/citation.cfm?doid=2808475.2808476.

[456] Zhan, Z., Xu, M., AND Xu, S. A Characterization of Cybersecurity Posture from Network Telescope Data. In: *Trusted Systems*. Ed. by Yung, M., Zhu, L., AND Yang, Y. Vol. 9473. tex.ids: zhan2014a, zhanCharacterizationCybersecurityPosture2015a. Springer International Publishing, 2015, 105–126. ISBN: 978-3-319-27997-8. DOI: 10.1007/978-3-319-27998-5_7. URL: http://link.springer.com/10.1007/978-3-319-27998-5%5C_7.

[457] Zhang, C., Patras, P., AND Haddadi, H. Deep Learning in Mobile and Wireless Networking: A Survey. *arXiv:1803.04311 [cs]* (Jan. 2019). arXiv: 1803.04311. URL: http://arxiv.org/abs/1803.04311.

[458] Zhang, J., ET AL. ProNE: Fast and Scalable Network Representation Learning. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, 4278–4284. ISBN: 978-0-9992411-4-1. DOI: 10.24963/ijcai.2019/594. URL: https://www.ijcai.org/proceedings/2019/594.

[459] Zhang, J., ET AL. On the Mismanagement and Maliciousness of Networks. In: *Proceedings 2014 Network and Distributed System Security Symposium*. tex.ids: zhang2014a, zhangMismanagementMaliciousnessNetworks2014a. Internet Society, 2014. ISBN: 978-1-891562-35-8. DOI: 10.14722/ndss.2014.23057. URL: https://www.ndss-symposium.org/ndss2014/programme/mismanagement-and-maliciousness-networks/.

[460] Zhang, M., ET AL. Network Diversity: A Security Metric for Evaluating the Resilience of Networks Against Zero-Day Attacks. *IEEE Transactions on Information Forensics and Security* 11, 5 (May 2016). tex.ids: zhang2016a, zhangNetworkDiversitySecurity2016a, zhangNetworkDiversitySecurity2016b tex.citation-number: 115 publisher: IEEE, 1071–1086. ISSN: 1556-6013, 1556-6021. DOI: 10.1109/TIFS.2016.2516916.

[461] Zhang, S., Zhang, X., AND Ou, X. After we knew it: empirical study and modeling of cost-effectiveness of exploiting prevalent known vulnerabilities across IaaS cloud. In: *Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14*. tex.ids: zhang2014b, zhangWeKnewIt2014a. ACM Press, 2014, 317–328. ISBN: 978-1-4503-2800-5. DOI: 10.1145/2590296.2590300. URL: http://dl.acm.org/citation.cfm?doid=2590296.2590300.

[462] Zheng, R., Lu, W., AND Xu, S. Active cyber defense dynamics exhibiting rich phenomena. In: *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security - HotSoS '15*. tex.ids: zheng2015a, zhengActiveCyberDefense2015a, zhengActiveCyberDefense2015b, zhengActiveCyberDefense2015c. ACM Press, 2015, 1–12. ISBN: 978-1-4503-3376-4. DOI: 10.1145/2746194.2746196. URL: http://dl.acm.org/citation.cfm?doid=2746194.2746196.

[463] Zhu, H., ET AL. Quality of experience and quality of protection provisions in emerging mobile networks [guest editorial. *IEEE Wireless Communications* 22, 4 (Aug. 2015). Citation Key: zhu2015a tex.citation-number: 59, 8–9.

[464] Zhu, Q., AND Rass, S. Game Theory Meets Network Security: A Tutorial at ACM CCS. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Jan. 2018). arXiv: 1808.08066, 2163–2165. DOI: 10.1145/3243734.3264421.

[465] Zimmermann, H. OSI reference model-the ISO model of architecture for open systems interconnection. *IEEE Transactions on communications* 28, 4 (1980), 425–432.

[466] Zonouz, S., AND Haghani, P. Cyber-physical security metric inference in smart grid critical infrastructures based on system administrators' responsive behavior. *Computers & Security* 39, PART B (Nov. 2013). Citation Key: zonouz2013a tex.citation-number: 3, 190–200.

[467] Zonouz, S., ET AL. Seclius: An information flow-based, consequence-centric security metric. *IEEE Transactions on Parallel and Distributed Systems* 26, 2 (Feb. 2015). Citation Key: zonouz2015a tex.citation-number: 118, 562–573.